Titre: Title:	Localisation mutuelle de plates-formes robotiques mobiles par vision omnidirectionnelle et filtrage de Kalman
Auteur: Author:	Vincent Zalzal
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Zalzal, V. (2005). Localisation mutuelle de plates-formes robotiques mobiles par vision omnidirectionnelle et filtrage de Kalman [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7757/

UNIVERSITÉ DE MONTRÉAL

LOCALISATION MUTUELLE DE PLATES-FORMES ROBOTIQUES MOBILES PAR VISION OMNIDIRECTIONNELLE ET FILTRAGE DE KALMAN

VINCENT ZALZAL
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
DÉCEMBRE 2005

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

LOCALISATION MUTUELLE DE PLATES-FORMES ROBOTIQUES MOBILES PAR VISION OMNIDIRECTIONNELLE ET FILTRAGE DE KALMAN

présenté par: ZALZAL Vincent
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. GOURDEAU Richard, Ph.D., président
M. COHEN Paul, Ph.D., membre et directeur de recherche
Mme PINEAU Joelle, Ph.D., membre

REMERCIEMENTS

Je tiens à remercier mon directeur, M. Paul Cohen, pour ses judicieux conseils et pour m'avoir donné la chance de travailler dans son laboratoire, le GRPR.

Je souligne également l'appui du *Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG)* qui a cru en moi et m'a octroyé une bourse de recherche.

Je remercie M. Vladimir Polotski pour ses explications à propos du filtre de Kalman et M. Richard Gourdeau pour son implantation du filtre en langage `C`.

J'ai apprécié travailler avec M. Sylvain Marleau à la création d'une librairie de filtre de Kalman étendu en langage `C++`. De la même façon, le temps passé conjointement avec M. Vincent Paquin et M. David Pinson pour travailler sur l'architecture logicielle *Acropolis* fut agréable.

Je remercie également tous les étudiants et employés du GRPR pour leur aide et leurs opinions, plus particulièrement, M. Hai Nguyen pour son aide lors de la sélection du senseur visuel, M. Jean Herbière pour son aide à la conception des points de repère, M. Thomas Gerbaud pour la conception du montage des senseurs sur la plate-forme mobile, M. Raphaël Gava pour la mise en place du réseau sans fil et M. Alexandre Fortin pour l'implantation d'une interface graphique accompagnant *Acropolis*.

Je remercie du fond du coeur Louise et André Zalzal, mes parents, pour leur soutien inconditionnel.

Enfin, j'ai une pensée pour Avril Lavigne, Babylon Circus, Caïman Fu, Jean Leloup, Les Cowboys Fringants, Manu Chao, Rammstein et Stefie Shock, pour leur musique qui m'a tenu compagnie tout au long de ce périple.

RÉSUMÉ

L'aptitude d'une plate-forme robotique mobile à se localiser est importante pour l'implantation de fonctions de navigation. D'autre part, la coopération entre plusieurs plates-formes dans le but d'accomplir des tâches concertées devient de plus en plus fréquente. La localisation mutuelle entre les plates-formes constitue donc un élément essentiel au maintien ou au contrôle d'une formation de robots.

L'objectif principal du projet est la conception algorithmique et l'implantation matérielle et logicielle d'un système de localisation mutuelle pour un ensemble de plates-formes mobiles. Ce système doit permettre, en temps réel, de mettre à jour et d'améliorer la localisation relative des plates-formes en déplacement. Ceci est possible dans la mesure où les plates-formes sont capables d'observer des points de repère communs au cours de leurs déplacements.

Les plates-formes sont munies d'un système d'odométrie afin d'évaluer la trajectoire parcourue, d'une caméra omnidirectionnelle afin de percevoir des points de repère autour d'elles et d'un système de communication sans fil permettant d'échanger des informations entre elles de manière fiable et ininterrompue.

On suppose que l'environnement dans lequel évoluent ces plates-formes est plat. Aucune autre hypothèse n'est émise à son sujet : l'environnement est inconnu et dynamique, c'est-à-dire que des obstacles peuvent se déplacer dans la scène. Cependant, les points de repère sont fixes par hypothèse, mais leurs positions sont initialement inconnues.

Le système d'odométrie est utilisé pour calculer le déplacement des plates-formes ; la localisation relative peut ainsi être mise à jour grâce au partage de ces informations entre les plates-formes. La détection de points de repère grâce à la caméra omnidirectionnelle est utilisée afin de corriger les erreurs de localisation.

Dans le but de concentrer les efforts de ce travail sur la conception d'une méthode de localisation mutuelle, le problème de détection et d'identification des points de

repère a été simplifié. Des balises cylindriques de couleur uniforme sont utilisées comme points de repère, chaque balise ayant une couleur unique.

Un filtre de Kalman étendu à dimension variable est utilisé pour mettre à jour l'estimation de la localisation relative des plates-formes et des points de repère. Il permet également d'intégrer la localisation des points de repère dans les images omnidirectionnelles à la correction apportée à l'estimation de la localisation relative. Des résultats de simulation ont permis d'étudier l'influence de différents facteurs sur la précision de l'estimation obtenue avec la solution proposée. Les résultats des tests avec des plates-formes réelles ont permis de valider la méthode : l'erreur de localisation d'une plate-forme dépasse rarement 20 cm pour un système bien calibré.

La méthode implantée est avantageuse à plusieurs niveaux. Elle n'exige aucun contact visuel entre les plates-formes, dans la mesure où celles-ci ne doivent pas s'observer mutuellement, mais doivent seulement pouvoir observer des points de repère communs. Seuls deux points de repère sont nécessaires au minimum pour que la méthode fonctionne convenablement. Aucune connaissance *a priori* de l'environnement ou de la position des points de repère n'est nécessaire. La méthode est peu coûteuse et rapide, et elle est suffisamment robuste pour s'adapter à des déplacements de points de repère, même si le modèle initial ne le prévoit pas.

La méthode de localisation relative mutuelle proposée pourrait servir de bloc de base à de futurs algorithmes décisionnels de robotique mobile. Entre autre, la localisation relative mutuelle est importante pour le maintien de formation ou l'organisation d'un rendez-vous entre plates-formes en milieu inconnu. La possibilité de se localiser mutuellement peut également permettre de couvrir une certaine zone de surveillance avec plus de fiabilité.

Une architecture logicielle évolutive d'intégration et de prototypage rapide a également été développée dans le cadre du projet. Cette architecture permet d'accroître la flexibilité et l'efficacité du développement informatique en robotique mobile en

facilitant l'intégration des diverses aptitudes des plates-formes et en favorisant la réutilisation du code. La méthode de localisation mutuelle développée a été implantée sur les plates-formes mobiles grâce à cette architecture logicielle.

ABSTRACT

The ability of a robotic mobile platform to locate itself is needed to implement navigation functionalities. Researchers now study cooperation between platforms to achieve tasks with teams of robots. Mutual localization between platforms is paramount to control robot formation.

The main goal of this research is to design and implement a mutual localization system for multiple mobile platforms, including creating the algorithm and designing the hardware and software needed. The system must be real-time and be able to update and correct relative localization of moving platforms. This is possible if the platforms are able to detect the same landmarks while moving.

The platforms are equipped with an odometry system to calculate their path of travel, an omnidirectional camera to detect landmarks around them, and a wireless communication system to continuously and reliably share data.

We suppose the environment in which the platforms are moving is flat. No other hypothesis is made about the environment : it is unknown and dynamic, which means that obstacles can move. However, landmarks don't move by hypothesis, but their initial position is unknown.

The odometry system is used to calculate the movement of platforms ; the relative localization can be updated thanks to data sharing between them. Landmarks detection using the omnidirectional camera allows to correct localization errors.

To concentrate design efforts on a mutual localization method, the problem of detecting and identifying landmarks has been simplified. Cylindrical landmarks of uniform color are used, each one having its own color.

A variable-dimension extended Kalman filter is used to update the relative localization estimation of both platforms and landmarks. It also allows to correct the estimation by using data from the detection and localization of landmarks in the omnidirectional images.

Simulation results allowed to study the influence of many different factors on the precision of the estimation. Results from tests with real platforms allowed to validate the method : localization error of a platform rarely goes beyond 20 cm for a well-calibrated system.

The implemented method has many advantages. No visual contact between platforms is needed : they must not detect each other visually, they must only be able to detect the same landmarks. Only two landmarks are necessary for the algorithm to behave correctly. No *a priori* knowledge of the environment or landmarks position is needed. It is a low-cost and efficient solution. It is also robust enough to adapt itself to landmark movement, even if it hasn't been included in the initial model.

The implemented mutual relative localization method could serve as a building block for the next decisional algorithms in mobile robotics. Relative localization is also needed for formation control or rendezvous planning of platforms in an unknown environment. Also, surveillance using many platforms could be more reliable by using a relative localization method like the one designed in this work.

An evolutionary software framework has been developed for this project. This framework increase software flexibility and development efficiency. In the field of mobile robotics, it allows easy integration of platforms abilities and promotes code reuse. The mutual localization method has been implemented on the real platforms under this framework.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES NOTATIONS ET DES SYMBOLES

i, j	Indice dénotant une plate-forme
k	Indice dénotant une itération
ℓ	Indice dénotant un point de repère
N	Nombre de plates-formes
M^k	Nombre de points de repère détectés à l'itération k
q_1, q_2, q_3	Paramètre du miroir hyperboloïdal
f	Distance focale (en pixels) de la webcam
Z_i $_{i\in\{a,b,\ldots\}}$	Hauteur du foyer du miroir par rapport au sol sur la plate-forme i
(x_{ij}^k, y_{ij}^k) $_{i,j\in\{a,b,\ldots\}}$	Position de la plate-forme i dans le repère de la plate-forme j à l'itération k
θ_{ij}^k $_{i,j\in\{a,b,\ldots\}}$	Orientation de la plate-forme i dans le repère de la plate-forme j à l'itération k
$\mathbf{p}_{ij}^k = (x_{ij}^k, y_{ij}^k, \theta_{ij}^k)$ $_{i,j\in\{a,b,\ldots\}}$	Pose de la plate-forme i dans le repère de la plate-forme j à l'itération k
$\check{\mathbf{p}}_{ij}^k = (\check{x}_{ij}^k, \check{y}_{ij}^k, \check{\theta}_{ij}^k)$ $_{i,j\in\{a,b,\ldots\}}$	Pose de la plate-forme i à l'itération k, dans le repère de la plate-forme j de l'itération $k-1$
d_i^k	Longueur du déplacement de la plate-forme i pour passer de la pose k à la pose $k+1$
ψ_i^k	Direction du déplacement de la plate-forme i (par rapport à θ_{ii}^k) pour passer de la pose k à la pose $k+1$
ω_i^k	Rotation appliquée à la plate-forme i pour passer de la pose k à la pose $k+1$
$\mathbf{p}_{\ell i}^k = (x_{\ell i}^k, y_{\ell i}^k)$ $_{\ell\in\{1,2,\ldots\}}$ $_{i\in\{a,b,\ldots\}}$	Position du point de repère ℓ dans le repère de la plate-forme i à l'itération k

$(r^k_{\ell i}, \phi^k_{\ell i})$ $\ell \in \{1,2,...\}$ $i \in \{a,b,...\}$	Position du pixel du point de repère ℓ, en coordonnées polaires centrées, dans l'image omnidirectionnelle de la plate-forme i à l'itération k
$D^k_{\ell i}$ $\ell \in \{1,2,...\}$ $i \in \{a,b,...\}$	Indique si la plate-forme i a détecté le point de repère ℓ à l'itération k
$\tilde{(\cdot)}$	Estimation *a priori*
$\hat{(\cdot)}$	Estimation *a posteriori*
$\bar{(\cdot)}$	Bruit
$\mathbf{x}^k$	Vecteur d'état à l'itération k
$\mathbf{u}^k$	Vecteur de commande à l'itération k
$\mathbf{w}^k$	Vecteur de bruit de processus à l'itération k
$\mathbf{z}^k$	Vecteur de mesure à l'itération k
$\mathbf{v}^k$	Vecteur de bruit de mesure à l'itération k
$\mathbf{f}$	Modèle du processus
$\mathbf{h}$	Modèle de mesure
n	Nombre de variables d'état
p	Nombre de variables de commande
q	Nombre de variables de bruit de processus
m	Nombre de mesures
t	Nombre de variables de bruit de mesure
$\mathbf{A}^k, \mathbf{W}^k, \mathbf{H}^k, \mathbf{V}^k$	Matrices jacobiennes à l'itération k
$\mathbf{P}^k$	Matrice de covariance de l'erreur d'estimation à l'itération k
$\mathbf{Q}^k, \mathbf{R}^k$	Matrices de covariance du bruit à l'itération k
$\mathbf{K}^k$	Gain de Kalman à l'itération k
$\mathbf{R}^k$	Matrice de rotation de la plate-forme a à l'itération k
$\mathbf{t}^k$	Vecteur de translation de la plate-forme a à l'itération k

Le chapitre 4 concernant le système de vision redéfinit certains symboles. Voici la liste de symboles ayant trait à ce chapitre en particulier.

(x_i, y_i)	Coordonnées cartésiennes d'un pixel i du plan image
(x_0, y_0)	Coordonnées cartésiennes du centre optique du plan image
(r, θ)	Coordonnées polaires d'un pixel du plan image
(R, θ, Z)	Coordonnées cylindriques d'un point de la scène
Z_0	Hauteur du foyer du miroir par rapport au sol
a, b, c	Paramètres de base du miroir hyperboloïdal
q_1, q_2, q_3	Paramètres du miroir hyperboloïdal dépendant de a, b, c
f	Distance focale de la webcam (en pixels)
N	Nombre de pixels d'une plage
S	Somme des carrés des erreurs entre les pixels d'une plage et la meilleure droite traversant la plage
Q	Critère de qualité d'une plage

LISTE DES ANNEXES

INTRODUCTION

Les robots manipulateurs et les plates-formes mobiles autonomes sont maintenant courants dans les chaînes de montage et certains sites d'opération. Il est légitime de croire qu'à moyen terme, de tels manipulateurs et des plates-formes mobiles seront de plus en plus utilisés *en groupe* afin de réaliser en coordination des tâches complexes.

Afin de bien coordonner ses mouvements dans une société de plates-formes, il est nécessaire que chaque plate-forme puisse se localiser relativement aux autres afin de se déplacer pour atteindre un objectif spécifique de formation. Afin d'améliorer les résultats d'une tâche accomplie en groupe, l'intercommunication est un facteur important. Puisque toutes les plates-formes ont pour objectif de se localiser mutuellement, elles doivent échanger entre elles des informations afin de rendre plus précise l'estimation de leur configuration relative. Deux types d'information semblent particulièrement pertinents à l'atteinte d'un tel objectif : la position d'un point de repère commun de la scène dans le champ visuel de chaque plates-forme et le déplacement local de chaque plate-forme. Ce mémoire porte sur la conception et le développement d'une méthode de localisation mutuelle basée sur ces informations et d'une architecture logicielle propice à son implantation.

La localisation mutuelle a des applications importantes et concrètes. Par exemple, il peut être désirable pour un groupe de plates-formes de se déplacer en maintenant une configuration relative fixe, en convoi, en « V », etc. Un autre cas où la localisation joue un très grand rôle est le problème de rendez-vous : deux plates-formes ne connaissant initialement ni leurs positions relatives, ni leur environnement doivent se rencontrer à un endroit indéterminé *a priori*. Afin de planifier l'itinéraire à suivre, on doit donc disposer d'une technique permettant d'obtenir, à partir de données

sensorielles, une estimation de la position d'une plate-forme par rapport à une autre. Ainsi, dans le cadre de l'exploration de surfaces planétaires, il est efficace d'utiliser simultanément plusieurs plates-formes. En communiquant sans cesse entre elles, elles pourraient améliorer non seulement l'estimation de leur localisation mutuelle, mais également explorer conjointement le site, acquérir et fusionner des données topographiques locales en une carte globale de la surface planétaire.

Le contexte étudié suppose des plates-formes munies d'odométrie, non pas d'un système de positionnement absolu externe tel que le GPS. Ainsi, il leur est impossible d'obtenir directement leur position dans un repère fixe. Les plates-formes naviguent dans un environnement inconnu, donc aucune carte n'est disponible. L'environnement est non structuré et dynamique, des obstacles pouvant se déplacer dans l'environnement.

Plusieurs approches peuvent être suggérées afin de répondre au besoin de localisation relative. À partir d'une estimation de la localisation relative initiale des plates-formes, il est possible de facilement tenir à jour leur configurations relatives en utilisant l'odométrie afin d'estimer les déplacements effectués et un système de communication pour échanger ces données. Cependant, l'odométrie, aussi précise qu'elle puisse être, engendre inévitablement des erreurs à long terme. Ces erreurs s'accumulent et font en sorte que l'estimation de la localisation relative des plates-formes diverge de la réalité.

Si les plates-formes sont munies de télémètres laser, elles peuvent également se localiser à l'intérieur d'une carte globale en mettant en correspondance un profil du télémètre avec un endroit de la carte. Ensuite, lorsque toutes les plates-formes se sont localisées à l'intérieur de la même carte, il est possible de déduire leur configuration relative. Cependant, puisque l'environnement est inconnu *a priori*, une telle carte globale n'existe pas et doit être construite pendant la navigation.

Le problème consiste alors à fusionner les cartes locales des plates-formes en une carte globale. Cette approche a quelques désavantages. La nécessité d'un télémètre laser augmente le coût de la solution. Il peut également être difficile d'utiliser un télémètre dans un environnement non structuré extérieur. Enfin, la localisation et cartographie simultanée ($SLAM$) à plusieurs plates-formes est un problème difficile.

Une autre solution, plus simple, au problème de localisation relative est la vision directe. En effet, si une formation doit être maintenue ou un rendez-vous doit être fait et que la première plate-forme détecte l'image de la deuxième grâce à un senseur visuel, alors il est possible de se diriger immédiatement dans la bonne direction. Toutefois, il est fort probable qu'en raison d'obstacles ou de dénivellations du terrain, la ligne visuelle séparant les deux plates-formes soit obstruée, éliminant ainsi la possibilité de poursuivre l'estimation de la position relative de la plate-forme. Cette solution impose donc trop de restrictions à l'environnement.

Une solution générale consiste à exploiter l'environnement. Puisque les plates-formes naviguent dans le même site, la probabilité qu'elles voient les mêmes objets est grande. Sans imposer que les plates-formes établissent un contact visuel direct, on peut supposer que les plates-formes voient en permanence des points de repère communs de l'environnement, même si l'identité et le nombre de ces points de repère communs varient au cours du temps en raison des déplacements. Cette approche est celle que nous avons choisie pour ce projet, car elle est générale et nécessite un équipement à faible coût (odométrie et senseur visuel).

L'approche préconisée est donc la détection et la localisation approximative de points de repère de l'environnement grâce à un système de vision à grand angle d'ouverture. Si plusieurs plates-formes peuvent détecter les mêmes points de repère, elles peuvent communiquer entre elles les positions de ces points dans leur champ visuel et exploiter cette information pour se localiser mutuellement. Dans cette

approche, il n'est pas nécessaire que la position des points de repère soit connue *a priori*, un aspect qui préserve considérablement la généralité du problème.

Une fois les mesures de position des points de repère acquises et partagées entre les plates-formes, une méthode mathématique doit être utilisée afin de fusionner ces données et générer une correction à apporter à l'estimation de la localisation relative mutuelle obtenue grâce à l'odométrie. La méthode utilisée dans le cadre du projet est le filtrage de Kalman, une méthode probabiliste qui a fait ses preuves dans le domaine de la robotique depuis plusieurs années.

L'interaction de plusieurs plates-formes hétérogènes exige une infrastructure informatique favorisant la flexibilité, l'évolutivité, l'intégration et la réutilisation. Une architecture logicielle évolutive d'intégration et de prototypage rapide a été développée dans le cadre de ce projet afin de subvenir à ce besoin et de faciliter l'implantation de la méthode de localisation mutuelle proposée.

La méthode développée propose deux principales innovations. La première est la possibilité pour plusieurs plates-formes de se localiser mutuellement sans besoin de carte de l'environnement ni de contact visuel direct entre plates-formes. La seconde innovation est le filtre de Kalman étendu à dimension variable, une variante du filtre non linéaire original permettant de faire varier le nombre de variables d'état au cours du temps.

Le mémoire comporte six chapitres. Le chapitre 1 présente une révue de la littérature au sujet de la localisation en général, et plus particulièrement de la localisation relative. Le chapitre 2 expose la problématique de façon formelle et décrit sommairement les étapes nécessaires de l'algorithme ainsi que l'architecture fonctionnelle du système développé pour ce projet. Le chapitre 3 explique la nécessité de développer une architecture logicielle évolutive qui facilite l'intégration de différents

modules logiciels et la réutilisation de ces modules pour un ensemble de plates-formes hétérogènes. Le chapitre 4 présente l'algorithme de vision développé dans le cadre du projet, incluant le choix du senseur visuel et un bref résumé de la théorie projective utilisée. Le chapitre 5 explique brièvement la théorie des filtres de Kalman étendus et présente le modèle de processus conçu pour l'estimation de la localisation relative mutuelle. Enfin, le chapitre 6 expose les résultats obtenus avec l'algorithme développé, aussi bien en simulation que lors d'expériences réelles.

CHAPITRE 1

REVUE BIBLIOGRAPHIQUE

1.1 Localisation d'une seule plate-forme mobile

Pour qu'une plate-forme puisse se rendre d'un point à un autre dans son environnement de façon autonome, elle doit être capable de se localiser de façon absolue dans le repère de cet environnement. La localisation a été et est encore un sujet très actif dans le domaine de la robotique autonome ; plusieurs techniques efficaces ont déjà été développées pour une seule plate-forme, chacune étant adaptée à des contraintes particulières.

Les techniques de localisation font souvent la simplification suivante : l'environnement est structuré, c'est-à-dire de géométrie connue, comme une suite de murs et de portes à angle droit (Clerentin et al., 2001; Drocourt et al., 1999). Ces environnements sont communs et offrent beaucoup d'applications commerciales : robots-concierges, robots de surveillance, robots distributeurs dans les édifices médicaux, etc. Le terme semi-structuré est maintenant plus fréquent (Cullen et al., 1999; Lallement et al., 1998) : il signifie que l'environnement est globalement connu, mais que les détails doivent être découverts par la plate-forme, y compris des obstacles dynamiques dans la scène. Le problème d'un environnement non structuré est plus difficile, mais a également été exploré par plusieurs chercheurs (Madhavan et al., 2002).

Plusieurs méthodes de localisation sont possibles. Les plus populaires sont basées sur la détection de points de repère dans l'environnement. Ces points peuvent être

des objets de géometrie et position connues *a priori*. Cela oblige alors l'utilisateur à installer des bornes spécifiques dans l'environnement afin que la plate-forme puisse les détecter et se retrouver grâce à leurs positions, qui doivent être précisément calibrées. Naturellement, cette méthode donne de très bons résultats, mais son manque de flexibilité et son coût d'installation élevé la rendent peu attirante.

D'autres auteurs préfèrent utiliser des points de repères de géométrie connue, mais sans en connaître la position dans l'environnement, ce qui est déjà plus flexible et évite le calibrage. Dans cette optique, il y a ceux qui ajoutent des bornes à l'environnement, comme des bandes réfléchissantes le long des corridors, par exemple. D'autres utilisent directement les particularités des environnements structurés et semi-structurés construits par l'homme : détection de lignes verticales, de lignes horizontales, de coins de murs (Drocourt et al., 1999; Clerentin et al., 2000; Cullen et al., 1999; Lallement et al., 1998).

Finalement, ceux qui travaillent dans un environnment non structuré et inconnu ne peuvent se fier à ces simplifications utiles. C'est le cas des plate-formes d'exploration d'environnement hostile, comme un environnement minier, une site planétaire ou un champ de mines. Dans ce cas, la détection est plus difficile et plus ambiguë. Une méthode qui connaît du succès est la détection de courbures maximales dans l'espace d'échelle (Madhavan et al., 2002).

Ces méthodes se fient à la capacité des plates-formes à bien localiser les points de repère. À cette fin, plusieurs senseurs peuvent être utilisés. Le senseur préféré dans les environnements structurés et semi-structurés est le télémètre laser (Madhavan et al., 2002; Thrun et al., 2000), qui permet d'obtenir non seulement l'angle du point de repère par rapport à la plate-forme, mais également sa distance relative. Toutefois, le télémètre est souvent restreint à visualiser les obstacles dans un seul plan de l'espace, ce qui limite beaucoup les types de points de repère détectables.

Les sonars, qui n'ont pas ce désavantage, ont beaucoup été utilisés en navigation, mais leur faible précision et leur lenteur en font de moins bons outils de localisation.

Les systèmes de vision sont également utilisés pour détecter des points de repère complexes (Rekleitis et al., 2001). Ces systèmes se distinguent également par leur aptitude à retourner la distance au point de repère (cas d'un système stéréo (Cullen et al., 1999)) ou la possibilité d'obtenir une vue de 360° (cas d'une caméra omnidirectionnelle (Dellaert and Stroupe, 2002; Spletzer et al., 2001; Kato et al., 1999; Barth and Ishiguro, 1994)). Finalement, certains auteurs ont modélisé un système à deux senseurs : télémètre laser et caméra omnidirectionnelle (Clerentin et al., 2001; Clerentin et al., 2000; Lallement et al., 1998).

Un problème plus difficile, nommé SLAM (*Simultaneous Localization and Mapping* — localisation et cartographie simultanée), a lui aussi été largement abordé (Chatila and Laumond, 1985; Dissanayake et al., 2001; Choset and Nagatani, 2001; Thrun et al., 2004). Dans ce contexte, l'environnement est inconnu et on désire le cartographier. Toutefois, pour pouvoir incorporer de façon incrémentale les données recueillies par les senseurs à une carte globale, il faut pouvoir localiser la plate-forme par rapport à cette même carte : c'est le problème de la poule et de l'oeuf. Il est facile de localiser la plate-forme dans une carte *a priori* de l'environnement et il est également facile de construire une carte globale si l'on connaît précisément la position de la plate-forme à chaque prise de données des senseurs. Cependant, lorsque la cartographie et la localisation doivent être effectuées en même temps, le problème devient plus difficile.

1.2 Localisation mutuelle de plusieurs plates-formes

Même si de bonnes solutions ont été trouvées pour le SLAM (Nüchter et al., 2005), un nouveau domaine a intéressé plusieurs chercheurs au cours de la dernière décennie : la localisation mutuelle dans le cas des systèmes multi-robots. En effet, certaines tâches peuvent être effectuées plus facilement et plus efficacement avec plusieurs plates-formes simples plutôt qu'une seule plate-forme complexe. Le SLAM multi-robots est un problème d'envergure, mais envisageable. En effet, les plates-formes elles-mêmes peuvent servir de points de repère pour d'autres plates-formes et les communications entre les différents acteurs du système permettent d'améliorer la précision des cartes globales obtenues.

Toutefois, dans le cadre des systèmes multi-robots, la localisation relative, c'est-à-dire la détermination de la configuration spatiale des plates-formes les unes par rapport aux autres, semble plus intéressante que la localisation absolue, c'est-à-dire la position des plates-formes dans un repère global de l'environnement exploré. Comme chaque plate-forme est en train de construire sa propre carte locale, il est possible qu'une d'entre elle ne puisse en localiser une autre dans sa carte, engendrant des problèmes supplémentaires. Ainsi, il est plus simple de localiser les plates-formes entre elles et se servir de cette information mutuelle pour améliorer la localisation absolue individuelle de chacune et permettre une fusion plus précise des données locales.

Les chercheurs tentent depuis quelques années de trouver de bons algorithmes de localisation mutuelle en raison du nombre croissant d'applications de systèmes multi-robots. En effet, le SLAM multi-robots pourrait servir à cartographier plus rapidement un endroit inconnu et potentiellement dangereux (Thrun et al., 2000), par exemple, une mine instable. Dans le contexte de la manipulation et du transport d'objets, il est envisageable d'utiliser plusieurs plates-formes pour pousser conjoin-

tement un objet et ainsi le déplacer sans user d'outillage spécial, tel une pince pour tenir l'objet (Spletzer et al., 2001). Ces types d'applications impliquent en général le maintien de formations, c'est-à-dire le maintien d'une certaine configuration relative entre plates-formes en mouvement, avec une bonne précision, dans un environnement inconnu et potentiellement adverse.

Les méthodes courantes de localisation mutuelle supposent souvent qu'il est toujours possible pour une plate-forme donnée de voir les autres plates-formes (Spletzer et al., 2001) (ce qui constitue une hypothèse fondée dans le cas de problèmes de maintien de formation, par exemple). D'autres supposent qu'une carte de l'environnement existe déjà, afin que toutes les plates-formes puissent s'y localiser de façon absolue, leur permettant ainsi de se localiser mutuellement. Cette hypothèse est plus sévère et réduit beaucoup la flexibilité de la méthode ; certains auteurs vont résoudre le problème en utilisant une plate-forme particulière pour faire un premier repérage de l'environnement (Thrun et al., 2000).

Quelques techniques ont déjà été développées pour obtenir une bonne localisation mutuelle. On dénombre principalement trois approches : géométrique, probabiliste et ensembliste. La première approche est la plus ancienne et la plus intuitive (Dellaert and Stroupe, 2002; Spletzer et al., 2001; Kato et al., 1999). Il s'agit de lister un ensemble de contraintes géométriques et de résoudre le système, la plupart du temps par la technique des moindres carrés. Comme exemple de contrainte, citons le fait de prendre les plates-formes trois par trois pour former des triangles, et imposer que la somme des angles aux sommets soit de 180° (Spletzer et al., 2001; Kato et al., 1999). L'approche géométrique peut impliquer beaucoup de calculs, car la complexité croît souvent exponentiellement avec le nombre de plates-formes et les algorithmes d'optimisation sont parfois assez lents, avec convergence non garantie. Le bon côté est que la méthode ne dépend pas alors des itérations précédentes. Ainsi, elle empêche l'accumulation d'erreurs et peut également servir à initier une

autre méthode, plus précise, qui elle dépend des itérations précédentes et a donc besoin d'un état initial.

Les méthodes probabilistes en sont un bon exemple. Ces techniques itératives permettent de localiser les plates-formes (parfois en incluant une région probable plutôt qu'un point de localisation (Thrun et al., 2000)) très précisément, en se fiant sur les valeurs obtenues aux itérations précédentes. Parmi toutes les méthodes, la méthode probabiliste par filtres de Kalman est la plus populaire, car elle donne d'excellents résultats, est relativement simple à implanter de façon récursive et a une excellente complexité d'exécution. L'inconvénient principal de ces méthodes est la nécessité d'initialiser convenablement l'algorithme, ce qui peut parfois nécessité une méthode différente ou ajouter des contraintes supplémentaires sur l'environnement.

Finalement, certains chercheurs ont récemment proposé un nouveau modèle prometteur : le modèle ensembliste (di Marco et al., 2003). Il s'agit, là encore, d'un algorithme récursif ayant de bonnes performances. Toutes les opérations sont effectuées sur l'ensemble des positions possibles de chaque plate-forme, sous forme d'unions, d'intersections, etc.. L'avantage principal de cette méthode est la possibilité d'obtenir des bornes sur l'estimation de la position des plates-formes, sous la forme de boîtes englobantes[1].

Naturellement, les algorithmes récursifs ont tendance à donner de meilleurs résultats, du fait qu'ils tirent profit des résultats précédents afin d'améliorer la précision des prochaines estimations. Toutefois, ils ont également quelques faiblesses. Puisque l'algorithme doit connaître l'itération précédente afin de calculer l'itération courante, il devient nécessaire de connaître l'état initial du système, ou au moins une estimation de celui-ci. On peut alors supposer que l'état initial est connu (ce qui peut être le cas dans certains contextes, comme par exemple si les plates-formes dé-

[1] Traduction de l'anglais « bounding boxes »

marrent toujours du même point à l'intérieur d'un bâtiment). Sinon, il faut un autre algorithme, non itératif celui-là, pour obtenir une estimation de la position initiale. C'est là que les techniques par contraintes géométriques évoquées précédemment ont toute leur utilité.

Parce que les techniques non itératives sont souvent moins précises, il peut y avoir une erreur initiale assez élevée. Or, la convergence des algorithmes utilisés n'est pas garantie, ce qui signifie que, dans certains cas, l'erreur peut augmenter avec le temps au lieu de s'estomper : c'est le problème des minima locaux. La convergence vers la bonne solution n'est assurée que si l'on se trouve dans le bassin d'attraction de ladite solution, ce qui explique qu'une estimation initiale erronée puisse causer des dégâts pour les filtres de Kalman, par exemple.

1.3 Filtre de Kalman

Il serait difficilement justifiable de présenter ici une revue bibliographique de l'utilisation du filtre de Kalman. Il est tellement répandu qu'un mémoire ne suffirait probablement pas à une telle revue. Toutefois, cette section parle de documents de référence pour l'implantation du filtre de Kalman étendu dans le cadre du projet.

Le texte de Kalman lui-même est disponible (Kalman and Bucy, 1960). Un texte introductif au filtre de Kalman permet d'avoir une approche intuitive à son utilisation (Welch and Bishop, 2003). Toutefois, l'implantation directe des concepts présentés dans cet article mène à un algorithme numériquement peu stable.

Dans le cadre du projet, la décomposition UDU (Bierman, 1977) a été utilisée afin d'améliorer la précision et la stabilité numérique du filtre de Kalman. Cette décomposition permet de calculer la « racine carrée » d'une matrice. Exprimer les formules du filtre de Kalman traditionnel sur les matrices décomposées permet de

conserver assurément la symétrie des matrices et leur propriété définie positive.

1.4 Architecture logicielle de prototypage rapide

L'interaction entre plates-formes multiples et hétérogènes exige une architecture informatique qui supporte cette multiplicité et cette diversité. Le désir de pouvoir utiliser le même logiciel sur différentes plates-formes, de pouvoir intégrer différents logiciels entre eux et de pouvoir simuler les plates-formes mène à la conception et à l'utilisation d'une architecture logicielle de prototypage rapide.

Avec les avancées technologiques continuelles, l'objectif d'une architecture informatique n'est plus d'améliorer l'efficacité de la machine, mais plutôt celle de l'humain lors des tâches de conception et développement de logiciels. La science informatique se penche maintenant davantage sur les concepts d'adaptabilité, de reconfiguration dynamique et d'interchangeabilité.

L'architecture logicielle peut n'être qu'un ensemble de spécifications conceptuelles à respecter, ou plutôt être un système concret d'implantation de logiciel facilement intégrable. Aussi, plusieurs paradigmes différents ont été explorés dans le domaine des architectures logicielles appliquées à la robotique. Il est possible de diviser en trois grandes catégories les architectures logicielles pour la robotique qui existent : les architectures à paradigme comportemental, les architectures à paradigme « Acquisition-Planification-Exécution » et les architectures ouvertes.

Le paradigme comportemental prône l'utilisation de systèmes délibératifs afin de chosir le meilleur comportement à adopter selon la situation actuelle et de l'objectif à atteindre. Par la suite, c'est le comportement qui, en fonction des données sensorielles choisira les actions à entreprendre. Des exemples d'architecture utilisant

ce paradigme sont *Saphira* (Konolige and Myers, 1998), *DAMN* [2] (Rosenblatt and Thorpe, 1995) et *ISR* [3] (Andersson et al., 1998). Afin de passer d'un comportement à l'autre, certaines architectures utilisent une machine à états finis (*ISR*) alors que d'autres utilisent un contrôleur à logique floue (*Saphira*). Peu importe le système, un point négatif de ces architectures est qu'un développeur ne peut décider de changer le mode de prise de décision : il doit choisir l'architecture adéquate au départ et s'en tenir à elle par la suite.

Saphira utilise un espace mémoire partagé afin de fusionner les données sensorielles et les rendre disponibles à tous les comportements : l'« espace de perception locale ». Au contraire, *DAMN* préfère un plus grand découplage : chaque comportement accède seulement aux données sensorielles nécessaires afin d'éviter des fusions inutiles.

Le paradigme « Acquisition-Planification-Exécution », de l'anglais « Sense-Think-Act » ou « Sense-Plan-Act », consiste en itérations successives et régulières où, en premier lieu, les données des senseurs sont colligées et fusionnées. Un plan à court terme est ensuite établi en fonction du but à atteindre. Ce plan est enfin mis en oeuvre et poursuivi pendant un intervalle de temps (habituellement très court), puis le cycle recommence. Le concept de comportement est absent de ce paradigme : l'action à entreprendre est choisie directement en fonction des données sensorielles. Il est à noter qu'un système « Acquisition-Planification-Exécution » peut être utilisé afin de constuire une architecture à paradigme comportemental.

La NASA a rédigé les spécifications d'une architecture logicielle usant d'un tel paradigme : *NASREM* (Albus, 1992). Il s'agit d'un système plutôt conceptuel, avec mémoire globalement accessible et trois étapes principales : fusion des données,

[2] Distributed Architecture for Mobile Navigation.
[3] Intelligent Service Robot.

modélisation de l'environnement (qui met à jour la mémoire globale) et décomposition des tâches (en séquences et en exécutions parallèles). Un autre exemple est *Player* (Gerkey et al., 2003), une architecture logicielle simple sous la forme d'un serveur de données sensorielles. L'avantage principal de ce système est la possibilité de simuler des plates-formes mobiles.

Les deux catégories précédentes présentent des architectures implantant un paradigme bien précis. Il est toutefois possible de concevoir une architecture logicielle qui ne contraint pas le développeur à utiliser un paradigme en particulier : il s'agit d'une architecture ouverte. Un tel système sert principalement à définir des schémas de communication entre des modules programmés par le développeur afin de faciliter l'intégration et la réutilisation.

Ces schémas peuvent être des spécifications formelles à respecter, comme dans le cas de *MARIE* [4] (Côté et al., 2004). À l'opposé, l'architecture ouverte peut combiner des modules fonctionnels générés semi-automatiquement grâce à une description formelle, comme c'est le cas pour *GenoM* (Fleury et al., 1997). Enfin, plusieurs architectures, telles que *OCP* [5] (Wills et al., 2001), définissent des modules par leurs entrées, leurs sorties et la couche d'abstraction à laquelle ils appartiennent.

1.5 Solution proposée

Ce projet consiste à développer une méthode de localisation mutuelle pour plusieurs plates-formes mobiles, en environnement inconnu et non structuré. L'approche probabiliste a été utilisée afin de calculer l'estimation de la localisation en se basant sur la détection et l'identification de points de repère de géométrie connue, mais

[4] *Mobile and Autonomous Robotics Integration Environment.*
[5] *Open Control Platform.*

de position inconnue. Les points de repère sont détectés à partir d'images omnidirectionnelles obtenues avec un système catadioptrique. La différence majeure de la méthode réside dans le fait qu'il n'est pas nécessaire que les plates-formes puissent se détecter, il suffit qu'elles détectent et identifient des points de repère communs.

Le filtre de Kalman étendu est utilisé afin de mettre à jour et de corriger la localisation. La décomposition UDU des matrices du filtre permet d'augmenter la précision de la méthode. L'algorithme implanté utilise un filtre à dimension variable, ce qui constitue un élément innovateur de la solution.

Une architecture logicielle évolutive d'intégration et de prototypage rapide a été développée dans le cadre du projet. Il s'agit d'une architecture ouverte, c'est-à-dire qui n'impose aucun paradigme particulier. Cette architecture propose quelques aspects innovateurs qui seront décrits plus loin dans ce mémoire.

CHAPITRE 2

ARCHITECTURE DU SYSTÈME

Ce chapitre introduit formellement le problème à résoudre (section 2.1) et énonce la solution retenue dans le cadre de ce projet (section 2.2). Une description sommaire accompagnée du schéma-bloc de haut niveau du système sera présentée, suivie d'une analyse fonctionnelle plus détaillée (section 2.3). Finalement, le matériel pour réaliser le projet sera décrit (section 2.4).

2.1 Problématique

Soit au moins deux plates-formes étant capables :
- de se mouvoir,
- d'évaluer la trajectoire qu'elles ont elles-mêmes parcourue,
- de communiquer entre elles sans fil,
- d'effectuer des calculs avec une performance suffisante et
- de voir, détecter, différencier et localiser approximativement certains points de repère de leur environnement.

Soit un environnement majoritairement plat, pouvant contenir des dénivellations, des objets mobiles et immobiles, dans lequel évoluent les plates-formes précédemment évoquées.

Supposons que les plates-formes ne connaissent pas *a priori* la configuration de l'environnement, mais que chaque plate-forme connaisse de manière approximative, dans le repère centré sur elle, la pose initiale des autres plate-formes. Finalement,

supposons qu'il n'est pas toujours possible pour chaque plate-forme de voir les autres plates-formes, mais que la communication entre elles n'est jamais interrompue.

Connaissant la localisation relative mutuelle initiale des plates-formes, il est simple de la mettre à jour en utilisant l'estimation des trajectoires parcourues par chacune des plates-formes. Toutefois, étant donné que le moyen utilisé pour calculer la trajectoire d'une plate-forme, en l'occurence, l'odométrie, engendre une erreur qui ne cesse de s'accumuler, la localisation relative mutuelle se détériorera avec le temps. Le problème posé est le suivant :

> Quel algorithme peut-on utiliser en temps réel afin que la localisation relative mutuelle soit mise à jour et que son estimation s'améliore plutôt que se dégrade ?

2.2 Solution proposée

En premier lieu, il faut trouver le moyen pour une plate-forme de recalculer constamment la pose des autres plates-formes à partir de leurs poses précédentes : les plates-formes doivent communiquer les informations de déplacement entre elles. Par la suite seulement sera-t-il possible de faire converger cette estimation vers la réalité en tenant compte de mesures prises par les plates-formes afin de donner de l'information supplémentaire sur la localisation relative mutuelle. Avant tout, définissons cette dernière formellement.

2.2.1 Localisation relative mutuelle

Afin de simplifier la notation et les calculs, un repère bidimensionnel sera utilisé. Cette simplification s'appuie sur l'hypothèse énoncée à la section 2.1 affirmant que le sol est principalement plat. Il demeure néanmoins que la généralisation au cas tridimensionnel ne pose aucun obstacle théorique particulier et pourrait permettre d'éliminer cette contrainte.

La localisation est la capacité qu'a une plate-forme de se positionner elle-même dans un environnement donné. La localisation est absolue si la plate-forme possède (ou construit simultanément) une carte de son environnement avec un repère global et est capable de donner sa position dans le repère de la carte. La localisation est relative si la plate-forme est capable de donner sa position par rapport à des points de repère, mobiles ou immobiles, de l'environnement.

La localisation relative est dite mutuelle si elle implique plusieurs plates-formes se localisant simultanément les unes par rapport aux autres, en partageant des informations. Pour une plate-forme donnée, la localisation relative mutuelle exprime la pose des autres plates-formes dans un repère centré sur elle-même. Par conséquent, le repère est mobile par rapport à l'environnement et ne sert qu'à exprimer la configuration relative des plates-formes, d'où son utilité pour le maintien de formations, par exemple.

La figure 2.1 définit graphiquement la localisation relative mutuelle, avec la notation qui l'accompagne. Les plates-formes se verront attribuer les lettres latines italiques minuscules consécutives : a, b, ..., souvent utilisées en indice. La pose d'une plate-forme i dans le repère centré sur la plate-forme j est représentée par le triplet $\mathbf{p}_{ij} = (x_{ij}, y_{ij}, \theta_{ij})$, où (x_{ij}, y_{ij}) est la position de la plate-forme et θ_{ij}, son orientation. Lorsque le contexte le permettra, l'indice inférieur j pourra être omis.

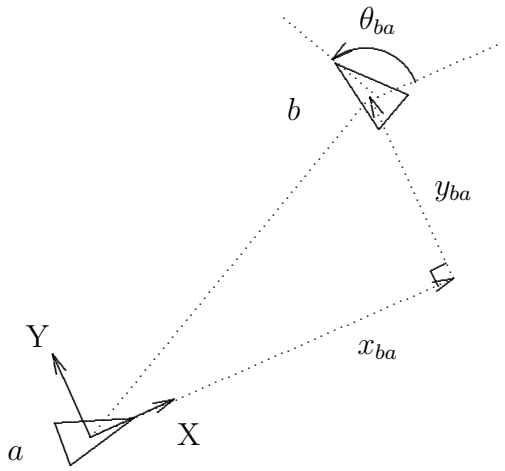


FIG. 2.1 Définition de la localisation relative mutuelle

Un repère centré sur une plate-forme implique que sa propre pose est $\mathbf{p}_{ii} = (0, 0, 0)$. L'axe des X du repère est confondu avec l'orientation $\theta_{ij} = 0$, en face de la plate-forme j,[1] signifiant que l'axe des Y pointe toujours à la gauche de la plate-forme j.

2.2.2 Mise à jour de la localisation

Afin de mettre à jour périodiquement la localisation, il est nécessaire, pour chaque plate-forme, d'estimer son propre déplacement en plus du déplacement des autres plates-formes. Par hypothèse, un système (odométrique, par exemple) permet à la plate-forme d'estimer son mouvement. Cependant, rien ne lui permet d'estimer par elle-même le mouvement des autres plates-formes, puisqu'il est possible qu'elles sortent de son champ de vision. Par conséquent, il faut recourir à la communication, qui, elle, est supposée ininterrompue.

Un intervalle de temps fixe sera déterminé *a priori* entre les plates-formes. Il s'agit de la période de rafraîchissement de la localisation. Nous appellerons cet intervalle

[1]Pour une plate-forme non holonome, l'axe des X est également confondu avec le vecteur vitesse.

un *quantum*, et l'action entreprise à chaque quantum, une *itération*.

Grâce à l'intercommunication, les quanta peuvent être synchronisés de telle sorte que toutes les plates-formes mettent leur localisation propre à jour en même temps. À cette fin, chaque plate-forme calcule sa trajectoire effectuée lors du dernier quantum dans son propre repère, puis la transmet aux autres plates-formes.

Lorsque la plate-forme a reçu les trajectoires de toutes les autres plates-formes, elle peut calculer la nouvelle pose de toutes les plates-formes, y compris elle-même, en appliquant un changement de repère et en effectuant le déplacement. Finalement, après avoir effectué tous les déplacements, il faut appliquer un dernier changement de repère global afin que le système d'axe soit de nouveau centré sur la plate-forme effectuant la localisation.

2.2.3 Amélioration de l'estimation

La procédure décrite à la sous-section précédente permet de tenir à jour la localisation, mais ne permet pas de la faire converger vers la localisation réelle. Pire encore, en raison des erreurs inhérentes à l'évaluation des trajectoires, la localisation estimée divergera de plus en plus. Il faut une autre mesure permettant de corriger les erreurs accumulées.

C'est là que se trouve la partie cruciale et innovatrice du projet. Par hypothèse, les plates-formes sont capables de détecter et de localiser des points de repère dans leur environnement, ne serait-ce qu'avec une faible précision.

Supposons les points de repère fixes dans l'environnement, sans toutefois imposer aux plate-formes d'en connaître la position *a priori*. Pour une plate-forme seule, localiser un point de repère lors d'une seule itération ne donne rien, car cela génère

autant d'équations que d'inconnues. Toutefois, si ce même point de repère est détecté lors d'itérations subséquentes, la position précédente du point de repère peut être utilisée afin d'améliorer la précision de la mesure. Si, après quelques itérations, il est possible de faire suffisamment confiance à la position estimée du point de repère, on peut aussi s'en servir pour améliorer l'estimation de la trajectoire effectuée lors d'un quantum. En effet, la position du point de repère devrait se préciser avec le temps, ce qui permettra de corriger les erreurs d'odométrie accumulées en basant le calcul de trajectoire par rapport au point de repère.

Maintenant, si plus d'un robot localise un même point de repère lors d'une itération, une nouvelle contrainte peut être ajoutée. En effet, une plate-forme peut envoyer à l'autre plate-forme la position estimée du point de repère dans son système d'axes. Après changement de repère, la seconde plate-forme se retrouve avec deux estimations de la position du point de repère : la sienne ainsi que celle de l'autre plate-forme, après changement de repère. Ces positions devant être identiques dans la réalité, cela permet d'apporter des corrections à la localisation relative entre les deux plates-formes. Ces corrections permettront de compenser pour les erreurs accumulées et de converger vers la localisation réelle, même si la localisation relative initiale était erronée.

Le problème à résoudre est de trouver le moyen d'incorporer ces mesures de manière à obtenir une correction optimale. Le choix que nous avons fait dans le cadre du projet est d'utiliser le filtrage de Kalman non linéaire, ou étendu. Grâce à ce filtre, il est possible, à partir d'un modèle d'évolution de l'état, de mettre à jour l'état (qui est la localisation relative mutuelle, dans le cadre de ce projet) et, du même coup, d'incorporer les mesures prises par les plates-formes, auxquelles sont attribuées des valeurs d'incertitude.

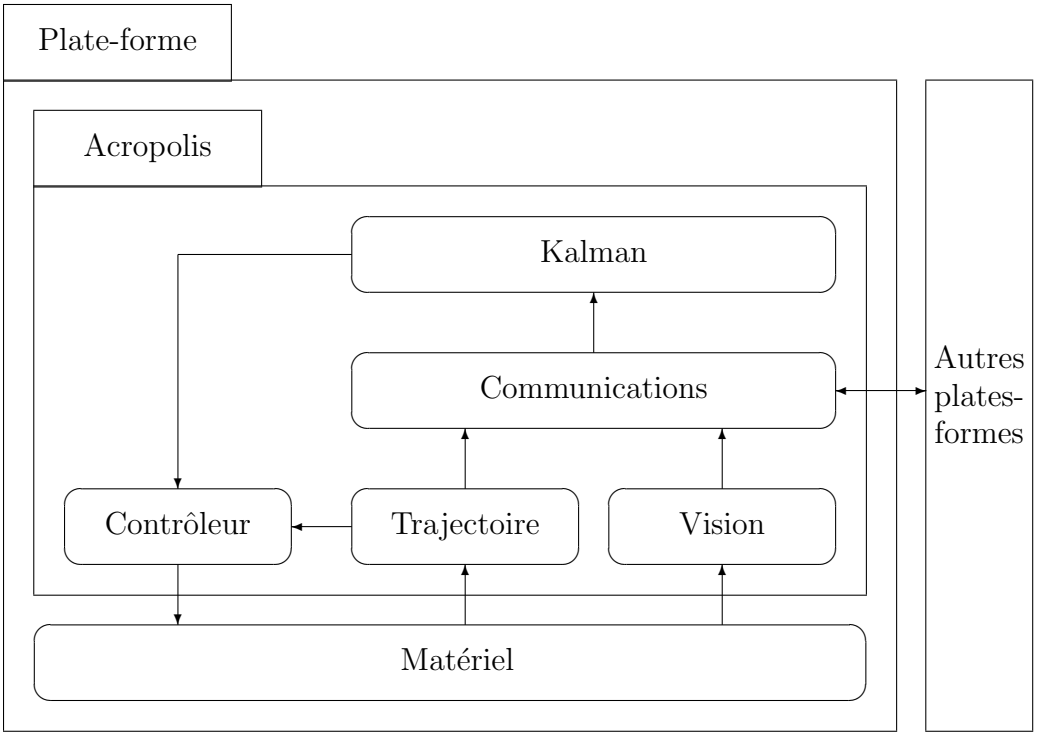


FIG. 2.2 Schéma fonctionnel du projet

2.3 Architecture fonctionnelle

La solution proposée dans le cadre de ce projet est réprésentée sous forme schématique à la figure 2.2. Y est représentée une plate-forme en particulier, avec son architecture logicielle (nommée *Acropolis*) et son architecture matérielle, communiquant avec les autres plates-formes via son module de communications. Chaque module fonctionnel sera maintenant décrit.

2.3.1 Module « Matériel »

Ce module s'occupe de gérer la communication avec les différents senseurs et actuateurs de la plate-forme, et ce, via des interfaces standardisées afin de découpler le reste du système des détails d'implantation matérielle. Dans le cadre du pro-

jet, la matériel comprend l'odométrie, le moteur de la plate-forme et la caméra omnidirectionnelle.

Pour une discussion du matériel utilisé, consulter la section 2.4. Pour de plus amples renseignements sur les techniques logicielles employées afin de découpler l'accès au matériel, voir le chapitre 3.

2.3.2 Module « Trajectoire »

Ce module s'occupe d'obtenir une estimation de la pose courante de la plate-forme. À la fin d'un quantum, les poses du début et de la fin de l'intervalle de temps sont comparées afin de générer le déplacement de la plate-forme lors de ce quantum, sous la forme d'une translation et d'une rotation.[2] Plus de détails sur les calculs effectués sont présentés au chapitre 5.

2.3.3 Module « Vision »

Le module de vision récupère une image à chaque quantum afin de détecter de potentiels points de repère dans l'environnement de la plate-forme. Les détails d'algorithmie et de contraintes imposées aux types de points de repère sont exposés au chapitre 4.

[2] L'analyse pourrait être tridimensionnelle, auquel cas le travail accompli par ce module nécessiterait les données d'un inclinomètre dans le module « Matériel » afin de connaître le tangage et le roulis de la plate-forme.

2.3.4 Module « Communications »

Ce module met à profit le travail investi dans l'architecture logicielle *Acropolis* en utilisant des mécanismes de communication flexibles et robustes entre les plates-formes. Ainsi, la détection des connexions et déconnexions des plates-formes est automatiquement gérée. Au-dessus de cette couche de base ont été implantés des mécanismes de « rassemblement et dispersion »[3] et de synchronisation.

2.3.4.1 Rassemblement et dispersion

Cette technique est couramment employée en programmation parallèle. Elle sert à répondre à un besoin fréquent dans ce domaine : chaque entité d'un système a une parcelle d'informations et désire obtenir les parcelles de toutes les autres entités. À cette fin, la technique opère en deux étapes :

Le rassemblement : chaque entité envoie sa parcelle d'informations à une entité particulière connue de tous. L'entité réceptrice rassemble toutes les parcelles pour former un paquet contenant toutes les informations.

La dispersion : ce paquet est ensuite envoyé à toutes les entités afin qu'elles connaissent les informations de toutes les autres.

Dans le cas qui nous intéresse, la parcelle d'informations contiendra la trajectoire parcourue par la plate-forme lors du dernier quantum ainsi que la position des points de repère détectés.

[3] Traduction libre de l'anglais « Gather/Scatter ».

2.3.4.2 Synchronisation

Il est important que toutes les plates-formes aient un quantum de même durée. Cependant, cela ne suffit pas. Lors de la mise à jour de la localisation relative mutuelle, il est impératif que toutes les plates-formes fournissent leur déplacement jusqu'au même moment dans le temps afin que la localisation représente l'état global du système à un temps donné.

Le mécanisme de synchonisation repose sur la barrière obligatoire de rassemblement et de dispersion expliquée précédemment. En effet, avant d'effectuer une nouvelle itération du filtre de Kalman, il est nécessaire d'avoir reçu les informations provenant de toutes les plates-formes. Ce moment précis (lorsque le paquet d'information est envoyé à tous[4]) permet à toutes les plates-formes de synchroniser leur horloge interne pour le prochain quantum.

2.3.5 Module « Kalman »

Le filtre de Kalman sert à mettre à jour la localisation relative mutuelle selon le modèle développé au chapitre 5 et à intégrer les mesures de vision afin de corriger cette localisation.

2.3.6 Module « Contrôleur »

Ce module est montré ici pour compléter le schéma global d'une plate-forme mobile autonome. Le but de faire de la localisation relative mutuelle est d'utiliser l'information résultante afin d'accomplir une tâche requérant la coopération de plus d'une

[4]Il s'agit ici de « broadcasting ».

plate-forme. Un tel contrôleur peut donc servir à faire un convoi de plates-formes ou à maintenir une formation en général.

Cet aspect sort du cadre de ce projet. Afin de générer les résultats nécessaires à la validation de la solution proposée, des contrôleurs très simples ont été implantés : un contrôleur envoyant une commande constante et un contrôleur manuel.

2.3.7 Module « Acropolis »

Acropolis est le nom donné à l'architecture logicielle implantée dans le cadre du projet. Bien qu'il s'agisse d'un aspect distinct du problème de localisation relative mutuelle, la complexité du projet nécessitait une infra-structure robotique flexible, robuste et évolutive. L'architecture est décrite plus en détail au chapitre 3.

2.4 Matériel utilisé

Afin de réaliser la solution proposée, deux plates-formes mobiles du laboratoire GRPR ont été utilisées. Cette section décrit les senseurs disponibles sur ces plates-formes ou ajoutés dans le cadre du projet.

2.4.1 Plates-formes ATRV-2 et ATRV-Mini

La figure 2.3 présente les deux plates-formes mobiles utilisées : l'*ATRV-2* et l'*ATRV-Mini*, toutes deux issues de la compagnie *iRobot*. Il s'agit de plates-formes à direction différentielle[5], c'est-à-dire que les quatre roues sont motrices, les deux roues d'un

[5]Traduction de l'anglais « skid-steer ».

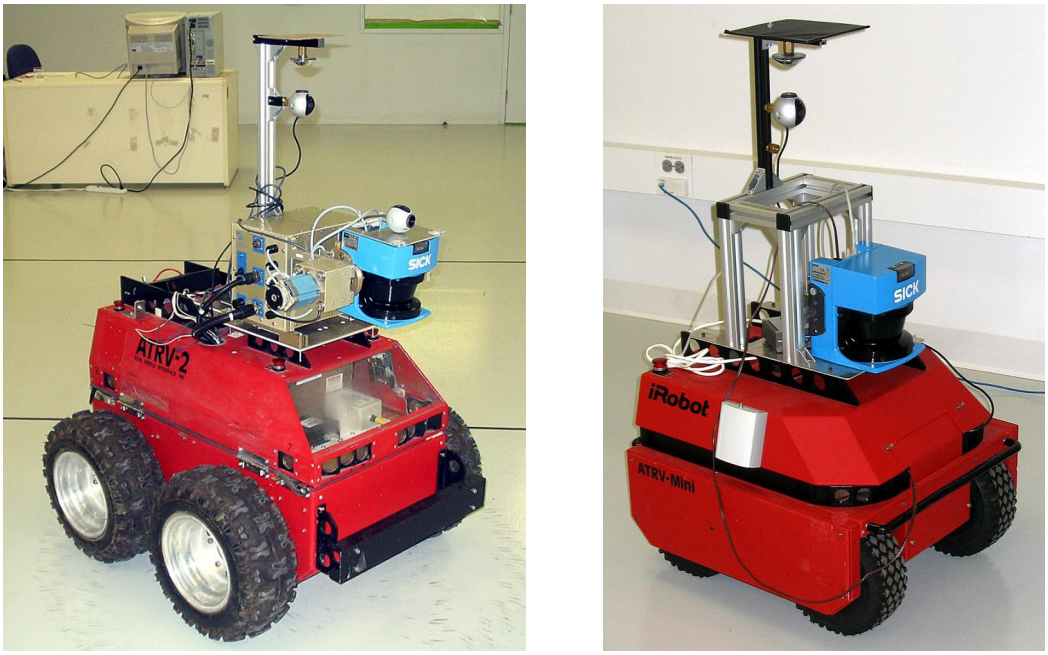


FIG. 2.3 Plates-formes ATRV-2 (à gauche) et ATRV-Mini (à droite), du GRPR

même côté tournant à la même vitesse. Ceci permet aux plates-formes de tourner sur elles-mêmes, si nécessaire.

L'odométrie intégrée permet d'estimer le déplacement des plates-formes avec une précision suffisante sur de courtes distances. Le contrôleur intégré permet de commander les plates-formes en vitesse. L'*ATRV-2* est muni d'un ordinateur à deux processeurs Xeon 3.2 GHz d'Intel avec 1 Go de mémoire vive. L'*ATRV-Mini*, quant à lui, possède un ordinateur avec un processeur Pentium III 1 GHz avec 128 Mo de mémoire vive.

Les deux ordinateurs fonctionnent avec le système d'exploitation Linux Fedora Core 3. Un système réseau sans fil permet la communication entre les plates-formes et le réseau du laboratoire GRPR.

FIG. 2.4 Caméra omnidirectionnelle montée sur les plates-formes mobiles

2.4.2 Caméra omnidirectionnelle

À la figure 2.4, on peut voir le montage utilisé comme appareil de vision : une caméra omnidirectionnelle. Grâce à un tel appareil monté sur chacune des plates-formes, il est possible de détecter simultanément, par traitement et analyse de l'image, des points de repère situés tout autour du robot.

Il s'agit en fait d'une webcam *QuickCam Pro 4000*, de *Logitech*, montée verticalement et faisant face à un miroir hyperboloïdal de la compagnie *NeoVision*, en République Tchèque. Les propriétés d'un tel montage, dit catadioptrique, sont discutées brièvement au chapitre 4 et sont détaillées à l'annexe I.

CHAPITRE 3

ARCHITECTURE LOGICIELLE

Des librairies propriétaires permettant de contrôler les plates-formes mobiles utilisées dans le cadre du projet étaient à la disposition du laboratoire au début du projet. Toutefois, le matériel informatique semblait désuet. Il était malheureusement impossible de mettre à jour l'ordinateur des plates-formes sans réinstaller la librairie de contrôle. Or, celle-ci ne fonctionne pas sur un système d'exploitation Linux récent.

Il existe une architecture logicielle libre nommée *Player* (Gerkey et al., 2003) (voir section 3.2) permettant de contrôler plusieurs plates-formes mobiles qui existent sur le marché, dont celles qui sont à la disposition du laboratoire GRPR. Toutefois, cette architecture comporte certains inconvénients.

Par conséquent, une architecture logicielle évolutive de prototypage rapide a été développée. Ce chapitre donne un aperçu du milieu dans lequel la nouvelle architecture logicielle est introduite. Tout d'abord, une liste des tâches fréquentes est établie afin de connaître les endroits où flexibilité, cohésion et découplage sont nécessaires. Les limites de l'utilisation seule de *Player* sont exposées. Finalement, la nouvelle architecture est présentée.

3.1 Analyse des besoins

Cette section donne un tour d'horizon de la situation d'un laboratoire de recherche en robotique. Certaines tâches doivent être effectuées fréquemment et il est souhai-

table de faciliter au maximum leur exécution grâce à une architecture logicielle de prototypage flexible et évolutive.

3.1.1 Environnement de travail

Un laboratoire de recherche en robotique consiste en un milieu de travail très volatile, et ce, à plusieurs niveaux. En premier lieu, le matériel utilisé au laboratoire change continuellement. En effet, du nouveau matériel est toujours nécessaire afin d'augmenter les capacités des plates-formes mobiles. De plus, le matériel existant devient rapidement désuet et le besoin de mise à jour est omniprésent si l'on veut rester à la fine pointe de la technologie. Enfin, les bris de matériel sont fréquents et le remplacement par des appareils équivalents mais différents est courant.

Afin d'exposer le travail accompli, un laboratoire invite plusieurs personnes à des démonstrations des capacités de ses plates-formes mobiles. Or, de telles démonstrations demandent parfois des changements majeurs de configuration matérielle des plates-formes ; l'interchangeabilité du matériel et des algorithmes de contrôle est un atout important.

Un étudiant typique travaillant dans un laboratoire de recherche y passe en moyenne deux années. Pendant ce temps, il y a une période d'apprentissage, une période de développement et une période de consolidation. Afin d'augmenter la période de développement, il est intéressant de réduire le temps d'apprentissage en augmentant l'efficacité des méthodes utilisées. La cohésion et le découplage sont des techniques appropriées afin de diviser l'ensemble du travail accompli au laboratoire en modules indépendants et autonomes. Un personne peut ainsi réutiliser les modules des autres sans se préoccuper du fonctionnement interne, ce qui fait gagner du temps à tout le monde.

Comme les anciens étudiants quittent rapidement le laboratoire et que de nouveaux arrivent, les problèmes d'intégration peuvent devenir cauchemardesques. En effet, sans cohésion et découplage, l'intégration de travaux exécutés par deux personnes différentes qui n'ont respecté aucun standard préétabli est difficile, sans compter que, souvent, il s'agit d'une troisième personne qui intègre les travaux antérieurs d'étudiants qui ne sont plus disponibles pour expliquer leurs projets.

Une architecture logicielle unifiée est nécessaire afin de donner des lignes directrices aux travaux de tous les étudiants. Ainsi, l'intégration s'en trouve facilitée et la réutilisation des travaux antérieurs permet le développement plus rapide de facultés intéressantes pour les plates-formes mobiles. Enfin, le transfert de connaissances y gagne en efficacité si la documentation des travaux est intégrée à l'architecture globale utilisée.

3.1.2 Tâches à accomplir fréquemment

La section 3.1.1 résume bien l'instabilité d'un laboratoire de recherche en robotique. De cette description, une liste d'actions couramment exécutées a été déduites afin de connaître les besoins en flexibilité de la nouvelle architecture logicielle. Voici les tâches à accomplir fréquemment, qui devraient donc être facilitées par l'utilisation de l'architecture logicielle :

1. Adapter les programmes disponibles à une autre plate-forme.
2. Changer l'ordinateur ou le système d'exploitation d'une plate-forme.
3. Changer un capteur pour un autre du même type, mais d'une marque différente.
4. Partager les données d'un capteur entre plusieurs programmes qui ne l'avaient pas prévu à l'origine.

5. Interchanger les capteurs entre les plate-formes *facilement* lors des démonstrations.
6. Intégrer des programmes différents qui n'avaient originellement pas été prévus à cette fin.
7. Tester un algorithme de façon répétitive avec les mêmes entrées.
8. Entraîner un nouvel étudiant à se servir du matériel du laboratoire.
9. Documenter l'utilisation des programmes et du matériel.

3.1.3 Objectifs : découplage et réutilisation

Il faut faciliter la réutilisation, l'intégration et l'abstraction du matériel afin que les tâches de la section 3.1.2 s'exécutent efficacement. À cette fin, les besoins en flexibilité suivants ont été identifiés :

Couche d'abstraction matérielle : il est primordial qu'une couche logicielle donne une interface de base commune à tous les capteurs et à tous les actuateurs. Ainsi, tous les capteurs peuvent être traités de manière uniforme grâce à l'interface de base commune et donc être initialisés, lus, activés, désactivés, etc., sans connaître le type de matériel en question. Une autre couche logicielle, spécialisée pour un type de capteur donné, est également nécessaire afin de fournir une interface spécialisée commune à tous les capteurs du même type, mais de marques différentes. Ceci évite d'avoir à modifier les algorithmes utilisant les données d'un capteur si la marque du capteur change. Changer de marque de caméra ne devrait pas influencer les algorithmes de vision utilisés dans le reste du programme, par exemple.

Encapsulation des facultés du système d'exploitation : il doit être facile de pouvoir adapter le code dans l'éventualité d'un changement de système d'exploitation. Mieux vaut donc ne jamais utiliser directement de faculté propre

au système d'exploitation et de les encapsuler dans une librairie. Ainsi, lorsqu'il faudra changer de système d'exploitation, seule cette librairie devra être adaptée au nouveau système.

Changement de configuration sensorielle simple : lors des démonstrations (et aussi pour que plusieurs personnes puissent travailler sur un nombre restreint de plates-formes mobiles), il est préférable de limiter au minimum le besoin de recompilation lors de changement du matériel sur la plate-forme. Idéalement, un fichier de configuration ou un programme avec interface graphique devrait se charger de décrire la liste des capteurs et actuateurs disponibles sur la plate-forme ainsi que la configuration initiale de ces capteurs et actuateurs (par exemple, le numéro du port série utilisé par un capteur).

Séparation entre les algorithmes et l'accès aux données : les algorithmes utilisés ne devraient jamais accéder directement aux données des capteurs ni contrôler directement les actuateurs. De cette façon, les entrées et les sorties d'un algorithme peuvent être modifiées à l'insu de l'algorithme lui-même. Cela permet d'ajouter des pré-traitements ou des post-traitements aux algorithmes sans les modifier et permet également à plusieurs algorithmes d'accéder aux mêmes données.

Déclenchement externe des algorithmes : un algorithme ne devrait pas se déclencher lui-même. Sinon, il devient impossible de synchroniser deux algorithmes entre eux. S'il existe un mécanisme externe pour déclencher les algorithmes (un signal d'horloge, par exemple), il est alors possible d'intégrer plus facilement les algorithmes et de les déclencher dans un ordre donné.

Encapsulation des blocs algorithmiques : tous les algorithmes devraient répondre à une interface standardisée, sous la forme d'entrées et de sorties. Cela permettrait d'interchanger facilement deux algorithmes accomplissant la même tâche, mais de manières différentes. Chaque bloc algorithmique se-

rait donc considéré comme une boîte noire isolée, avec des entrées, des sorties et un mode de déclenchement externe, sans possibilité d'interférer avec d'autres blocs algorithmiques autrement que par ses sorties. Il ne resterait plus qu'à interconnecter les entrées et les sorties des différents blocs algorithmiques afin d'obtenir un programme complet et flexible, à l'instar d'un circuit électronique utilisant plusieurs circuits intégrés à petite et moyenne échelle.

Intégration de tâches fréquentes dans l'architecture : il est probable qu'un système multi-robots nécessite un moyen de communication entre les robots. Plutôt que chaque développeur programme un nouveau schéma de communication, il devrait n'y avoir qu'un seul mode de communication universel entre les robots, robuste et facile à utiliser. De la même façon, des tâches courantes, comme le parsage d'expressions, devraient être bien programmées une fois pour toute, intégrées à l'architecture logicielle et réutilisées par la suite.

Intégration de la visualisation dans l'architecture : la plupart des blocs algorithmiques produisent un résultat qui ne peut être apprécié que par la visualisation à l'écran. Encore une fois pour éviter la répétition du code et pour promouvoir la facilité d'intégration, une interface graphique générale devrait être développée afin de pouvoir gérer chaque bloc algorithmique actuellement en service, et la communication avec cette interface graphique devrait être intégrée à la nouvelle architecture logicielle.[1]

Possibilité de simuler le fonctionnement de la plate-forme : il est beaucoup plus simple, en phase de développement, de tester les algorithmes en simulation. Cela permet, en outre, de désengorger l'accès à la plate-forme mobile et diminue le temps nécessaire pour tester les algorithmes. Toutefois, il faut que le passage du programme de simulation au programme réel ne requiert qu'un

[1] Ce point n'a pu être exploré en détail par manque de temps, mais fait partie des améliorations à apporter au système.

minimum d'effort pour en valoir la peine.

Evolutivité : il doit être facile d'ajouter un nouveau pilote de matériel ou un nouveau bloc algorithmique, et ce, sans nécessairement tout recompiler l'architecture.

3.2 *Player*

Afin de répondre en partie aux besoins énoncés à la section 3.1.3, le logiciel libre *Player* a été conçu par l'équipe de robotique de la *University of South California* (Gerkey et al., 2005). *Player* est un serveur utilisé pour contrôler une plate-forme mobile. Il procure une interface simple aux capteurs et actuateurs de la plate-forme par le biais du réseau IP. Il agit principalement comme couche d'abstraction matérielle, donnant à tous les capteurs et actuateurs une interface commune. Cette section parle plus en détails de *Player*, de ses avantages et de ses limites.

3.2.1 Principes de fonctionnement

La fonction de *Player* peut se résumer ainsi : le serveur rassemble les données provenant des capteurs, les envoie par TCP/IP via une interface standardisée aux clients qui les utilisent en fonction de l'application, puis renvoient des commandes là aussi standardisée au serveur, qui les appliquent aux actuateurs. La figure 3.1 montre les entités du système. Une plate-forme mobile contient nécessairement du matériel avec lequel il doit être possible de communiquer grâce à l'ordinateur à bord de la plate-forme. Chaque capteur ou actuateur a son protocole de communication propre, qui est géré par le serveur *Player*. Celui-ci s'occupe d'offrir une interface TCP/IP standard aux différents clients, qui peuvent être à bord ou non.

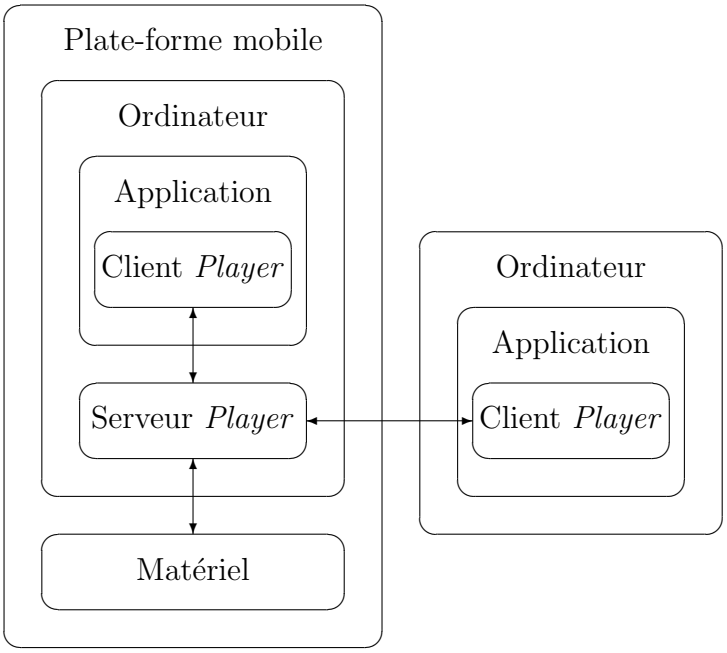


FIG. 3.1 Schéma de communication de *Player*

Afin d'être évolutif, *Player* gère chaque matériel en utilisant un pilote, comme le montre la figure 3.2. Chaque pilote est en fait une librairie dynamique compilée en-dehors de *Player*, ce qui facilite beaucoup le développement sans sacrifier la performance du système.

Chaque pilote s'exécute dans son unité d'exécution[2] propre afin de bien découpler les modules entre eux. Un gestionnaire de pilotes, lui-même dans une unité d'exécution distincte, s'occupe d'instancier les pilotes nécessaires selon le matériel décrit dans un fichier de configuration. Ceci permet de changer le matériel sur le robot et d'exécuter une même application en ne changeant que le fichier de configuration décrivant le matériel : aucune recompilation n'est nécessaire.

Le gestionnaire de pilotes s'occupe également des connexions et déconnexions des clients désirant obtenir des données provenant des capteurs et envoyer des com-

[2]Traduction de l'anglais « thread ».

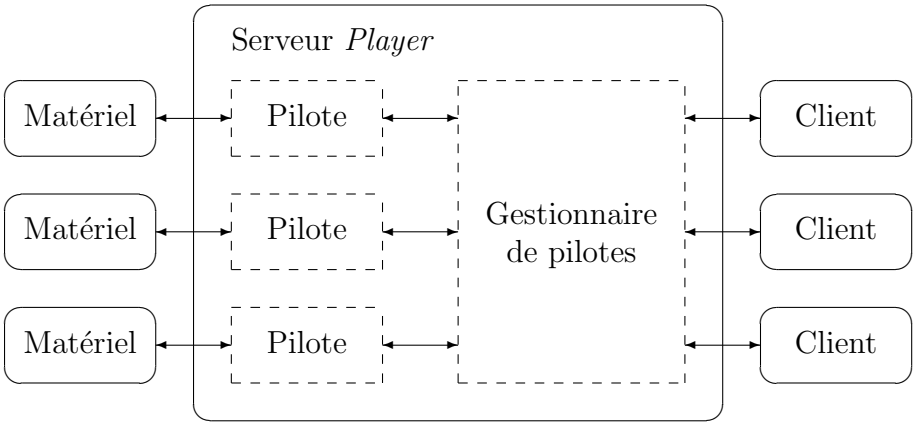


FIG. 3.2 Schéma fonctionnel de haut niveau de *Player* ; chaque boîte tiretée représente une unité d'exécution

mandes aux actuateurs.

Afin de simplifier au maximum l'écriture d'un nouveau pilote (tâche nécessaire lors de l'installation d'un nouveau capteur ou actuateur), *Player* fournit l'infrastructure gérant les problèmes de synchronisation, de protection de données par sémaphores et autres détails d'implantation (figure 3.3). Le programmeur n'a qu'à implanter le protocole de communication avec le nouveau matériel, récupérer les données, les mettre sous la forme standard selon le type de matériel et remplir un tampon de données. De la même façon, un tampon de commande est rempli par le gestionnaire de pilotes lorsqu'un client désire contrôler un actuateur. Finalement, un mécanisme de requêtes et réponses via des files permet de changer la configuration du matériel ou d'obtenir des informations sur cette même configuration.

3.2.2 Avantages

Utiliser *Player* comporte beaucoup d'avantages, d'où sa popularité dans les universités et centres de recherche en robotique (Brooks et al., 2004). En voici un aperçu :

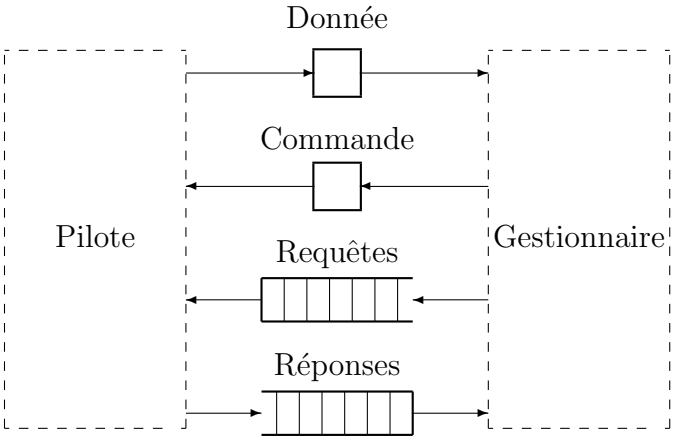


FIG. 3.3 Schéma fonctionnel de bas niveau de *Player*

Bonne abstraction matérielle : chaque capteur ou actuateur est encapsulé dans un pilote, et chaque module de contrôle de matériel a une interface commune, permettant un traitement uniforme et l'interchangeabilité des capteurs.

Configuration par fichier : pour changer le matériel d'une plate-forme mobile ou la configuration initiale des capteurs et actuateurs, il n'est aucunement nécessaire de recompiler quoi que ce soit. Il faut simplement éditer un fichier de configuration.

Encapsulation des blocs algorithmiques : ces blocs sont traités comme du matériel virtuel. Du point de vue du développeur, il n'y a pas de différence entre un télémètre laser et un détecteur de contour.

`playerv` **:** un visualisateur est fourni avec *Player*. Celui-ci permet de connaître l'environnement d'une plate-forme mobile, vu par ses capteurs, et permet d'agir directement sur les actuateurs de la plate-forme. Il s'agit en fait d'un client *Player* avec une interface graphique.

`stage` **:** un simulateur est fourni avec *Player*. Celui-ci permet de bien tester tous les algorithmes sans changer une seule ligne de code ou même recompiler le client *Player*. L'idée est de remplacer tous les pilotes que *Player* utilise pour contrôler le matériel par des pilotes de simulation, ce que `stage` fait.

Evolutivité : il est facile d'ajouter un nouveau pilote de matériel ou un nouveau bloc algorithmique, et ce, sans nécessairement tout recompiler l'architecture.

3.2.3 Limites

Malgré la liste impressionnante d'avantages de *Player* et la liste tout aussi impressionnante d'utilisateurs de cette librairie, il a été décidé que *Player* ne pouvait constituer une architecture logicielle complète pour plates-formes mobiles. En effet, l'abstraction des blocs algorithmiques n'est pas très développée (ce n'était d'ailleurs pas le but de *Player*). Voici les principales raisons qui nous ont poussés à l'élaboration d'une architecture complète :

Séparation entre les algorithmes et l'accès aux données : dans *Player*, les capteurs et les blocs algorithmiques se retrouvent tous au même niveau. Il n'y a pas d'architecture arborescente. Ainsi, chaque bloc algorithmique ayant besoin de données sensorielles va les chercher directement là où elles se trouvent. Ceci empêche deux bloc d'accéder aux données des senseurs simultanément (ce doit être fait dans le client).

Auto-déclenchement des algorithmes : c'est l'algorithme lui-même qui décide quand il doit s'exécuter. Ceci enlève une certaine flexibilité, entre autres, cela complique l'intégration de deux algorithmes qui n'avaient pas initialement été prévus à cette fin : il devient impossible de les séquencer, par exemple.

Structure linéaire : mettre tous les blocs, sensoriels et algorithmiques, sur un même niveau revient à simplifier à outrance un système robotique. En effet, cela voudrait dire qu'aucun algorithme ne dépend de la sortie d'un autre algorithme, à moins d'introduire un fort couplage entre les deux modules.

Structure fixe à l'exécution : puisqu'il n'y a pas d'arborescence, il n'y a nul besoin de pouvoir modifier les interconnexions entre les différents blocs al-

gorithmiques au moment de l'exécution. Or, cette possibilité est importante si l'on veut incorporer de l'intelligence artificielle aux plates-formes mobiles, afin qu'elles puissent changer de comportement selon certaines conditions préétablies, par exemple.

Il est à noter que ces limites se réfèrent à la version 1.5 de *Player*. La version 1.6, qui est apparue après le développement d'*Acropolis* décrit à la section suivante, semble pallier quelques-uns de ces problèmes. Entre autre, une forme de structure arborescente y est implantée. Le problème d'auto-déclenchement demeure bien réel cependant.

3.3 *Acropolis*

Les limites exposées précédemment ont été jugées inacceptables. Toutefois, pourquoi réinventer la roue ? *Player* fait déjà du bon travail à plusieurs niveaux, il suffit de combler les lacunes. C'est pourquoi un considérable effort de développement a été investi, dans le cadre de ce projet, dans l'élaboration de l'architecture *Acropolis*, qui englobe *Player* afin de gérer le matériel.

Les principes de base du fonctionnement d'*Acropolis* seront tout d'abord exposés, suivis des objectifs de flexibilité atteints grâce à la nouvelle architecture logicielle. Pour terminer, l'interface graphique *AcroBuilder* développée par M. Alexandre Fortin sera présentée.

3.3.1 Principe

Il est désirable d'isoler les algorithmes les uns des autres afin qu'ils ne s'influencent seulement que par une interface normalisée. C'est la base de plusieurs conseils de

programmation courants. Par exemple, l'utilisation de variables globales dans un programme est déconseillée, car il est toujours possible que deux algorithmes se fient sur la valeur de cette variable, mais ne soient pas au courant que tous deux la modifient simultanément. En isolant les algorithmes, il devient plus facile de les réutiliser et les intégrer pour faire des algorithmes plus complexes.

Un domaine qui a pleinement réussi l'exploit du découplage des algorithmes est l'électronique à moyenne échelle. Il est possible de se procurer des circuits intégrés exécutant une tâche bien précise, comme un compteur ou un registre à décalage, et de les intégrer en les interconnectant, sans se préoccuper du fonctionnement interne de ces circuits. Ceci est possible car les circuits sont autosuffisants ; ils ne dépendent pas d'autres circuits pour fonctionner.

Il y a une exception : le signal d'horloge n'est pas intégré aux circuits. Cependant, c'est un avantage, et non un inconvénient, car cela permet de contrôler le moment où le circuit va démarrer son exécution. Finalement, les entrées et sorties de ces circuits sont normalisées afin d'être compatibles entre elles (prenons pour exemple la technologie TTL).

Il apparaît que l'électronique à moyenne échelle a atteint plusieurs des objectifs recherchés. Il paraît donc raisonnable d'essayer de mimer le comportement de circuits intégrés en logiciel. C'est le principe de base de l'architecture logicielle *Acropolis*.

Chaque algorithme logiciel, pour être indépendant des autres, est isolé dans un module compilé séparément sous la forme d'une librairie dynamique. Chacune de ces librairies agit donc comme un circuit intégré et l'algorithme qui y est implanté s'exécute dans sa propre unité d'exécution.

Aucun module ne communique directement avec les autres afin de ne pas introduire de couplage inutile. De la même façon, aucun module ne traite directement avec le

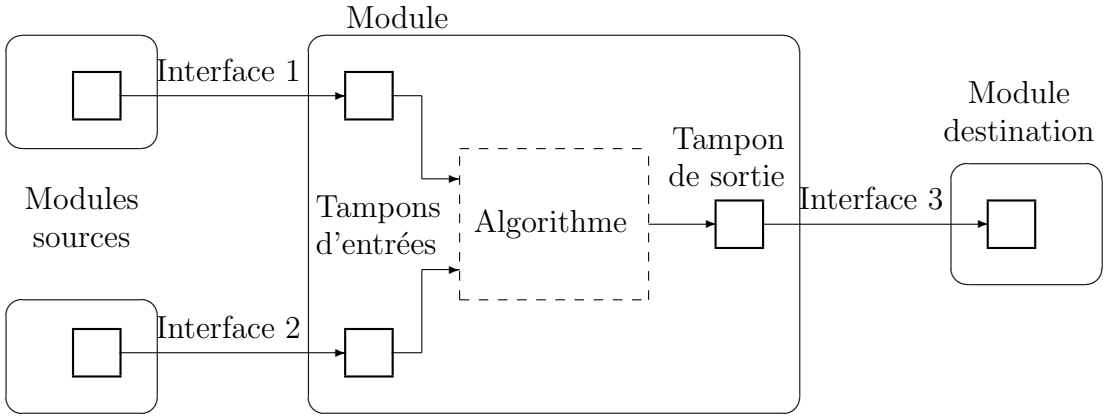


FIG. 3.4 Structure d'un module

matériel de la plate-forme mobile afin que les modules puissent être réutilisés avec du matériel différent, mais équivalent.

Toutefois, il est nécessaire d'avoir une communication minimale entre les modules afin d'accomplir une tâche quelconque, sans toutefois coupler les modules entre eux. C'est l'équivalent logiciel d'une connexion électronique.

Chaque module est représentée comme une boîte noire, ayant un certain nombre d'entrées et de sorties. Chacune d'entre elles est déclarée d'un certain type normalisé afin de connaître l'information qui entre ou sort du module.

Comme le montre la figure 3.4, chaque module possède des tampons d'entrées et de sorties afin de supporter le transfert d'informations d'un module à l'autre. Une connexion revient à indiquer quel tampon de sortie sera copié dans quel tampon d'entrée. Naturellement, l'architecture logicielle gère par elle-même les copies, et la notion même de tampons est invisible à l'utilisateur d'*Acropolis*. La gestion des mécanismes de protection de mémoire via des sémaphores est également supportée automatiquement par l'architecture.

Reste à spécifier le mode de déclenchement de l'algorithme. Afin de pousser l'ana-

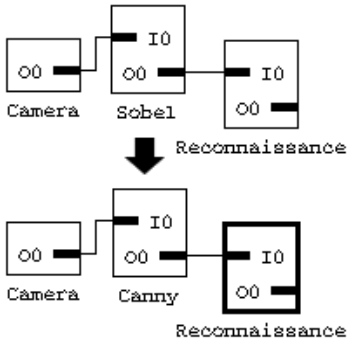
logie avec un circuit électronique davantage, un module est déclenché en général lorsqu'une de ses entrées change. Le module déclare lesquelles de ses entrées sont *sensibles*, c'est-à-dire susceptibles de déclencher l'exécution du module si les données du tampon correspondant changent.

Prenons l'exemple d'une porte ET en circuit intégré. La sortie du circuit change dès qu'une de ses entrées change : c'est un circuit asynchrone. Le même comportement est obtenu logiciellement en construisant un module sensible à toutes ses entrées. À l'inverse, prenons le cas d'un microcontrôleur, qui exécute une instruction à chaque coup d'horloge, circuit typiquement synchrone. Un résultat similaire peut être obtenu avec un module ayant une seule entrée sensible de type « Horloge ».

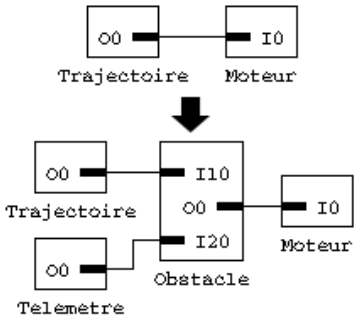
Avec ces concepts réunis, il est possible de concevoir un « circuit informatique » réalisant une tâche en réutilisant des modules de base, comme ceux montrés à la figure 3.5.

La flexibilité d'une telle structure est considérable. Il est possible d'échanger un algorithme pour un autre s'ils ont les mêmes entrées et sorties, par exemple, remplacer un détecteur d'arêtes de Sobel par un filtre de Canny (figure 3.5(a)). Il est possible d'intercaler un algorithme entre deux autres pour effectuer un pré-traitement ou un post-traitement, par exemple, ajouter un module d'arrêt d'urgence s'il y a un obstacle imprévu lors d'un suivi de trajectoire (figure 3.5(b)). Grâce au contrôle du déclenchement, il est possible de synchroniser des algorithmes entre eux. Par exemple, il peut être nécessaire qu'à chaque cinq secondes, une plate-forme effectue une rotation, puis prenne une photographie après avoir tourné (figure 3.5(c)).

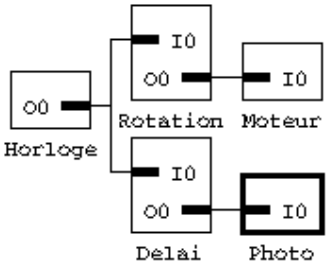
Grâce à la flexibilité de l'informatique, il est également possible de faire des choses qui seraient difficilement réalisables électroniquement. Une de ces aptitudes est la généricité (figure 3.5(d)). Il s'agit, pour un module, d'avoir des entrées et sorties



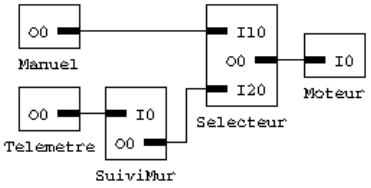
(a) Échanger



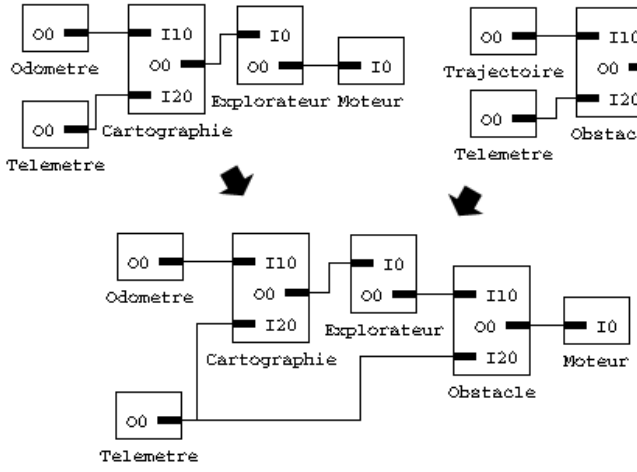
(b) Intercaler



(c) Synchroniser



(d) Généricité



(e) Intégrer

FIG. 3.5 Flexibilité d'*Acropolis*

de types connues seulement à l'exécution. Naturellement, les actions possibles sont restreintes, mais un même module peut dès lors être réutilisé pour différents types. Un sélecteur est un bon exemple : on peut choisir la commande à envoyer aux moteurs selon le mode de contrôle désiré de la plate-forme, ou choisir, grâce au même module, l'objectif à atteindre avec un planificateur de trajectoire.

Finalement, il devient aisé d'intégrer deux programmes. Par exemple, des modules de cartographie et d'évitement d'obstacles peuvent être combinés en un seul circuit pour créer une plate-forme capable de cartographier, tout en évitant les obstacles (figure 3.5(e)).

3.3.2 Objectifs atteints

L'architecture *Acropolis*, jumelée à *Player* pour gérer le matériel d'une plate-forme mobile, permet d'atteindre pratiquement tous les objectifs fixés à la sous-section 3.1.3.

3.3.2.1 Couche d'abstraction matérielle

Player est utilisé pour faire abstraction du matériel précis utilisé. Ainsi, pour passer d'une configuration matérielle à l'autre, il suffit de modifier le fichier de configuration passé à *Player*, aucune recompilation n'est nécessaire. Seules différentes librairies dynamiques seront chargées à l'exécution.

Si l'on veut ajouter le support d'un nouveau matériel dont l'interface existe déjà dans *Player*, il est seulement nécessaire de programmer un nouveau pilote et de le compiler (à l'extérieur de *Player*). Du point de vue d'*Acropolis*, aucune modification au schéma de connexions ni aucune compilation n'est nécessaire.

3.3.2.2 Encapsulation des facultés du système d'exploitation

Afin de faciliter l'adaptation éventuelle d'*Acropolis* à un autre système d'exploitation (temps réel, notamment), aucun appel direct aux facultés du système d'exploitation n'est présent dans le code. Une librairie dynamique encapsulant tous les appels dans des fonctions ou des classes a été conçue pour régler ce problème. *Acropolis* utilise cette librairie ; pour adapter *Acropolis*, il suffit de réécrire la librairie sans en modifier l'interface.

3.3.2.3 Séparation entre les algorithmes et l'accès aux données

Le principe même de connexions découplent les modules entre eux : la provenance des entrées d'un algorithme dépend directement du schéma de connexions. Afin d'accéder aux données en un point unique, des modules sont en fait des clients *Player* récupérant les données des capteurs et les mettant disponibles sur une sortie pour d'autres modules.

3.3.2.4 Déclenchement externe des algorithmes

Pour son bon fonctionnement, un module déclare son mode de déclenchement (quelles entrées sont sensibles). Toutefois, un module ne décide pas qui le déclenche, comme dans un circuit électronique. Le concept d'horloge permet de synchroniser des modules entre eux et même de pousser l'analogie du circuit électronique jusqu'à faire des diviseurs d'horloge afin qu'un algorithme s'exécute deux fois plus souvent qu'un autre, par exemple.

3.3.2.5 Encapsulation des blocs algorithmiques

Puisqu'un module est l'instance d'une classe appartenant à une librairie dynamique et obtenant ses entrées uniquement via une copie dans ses tampons d'entrées, il n'y a aucun moyen pour un bloc algorithmique d'interférer avec un autre. Un module ne peut pas changer les variables d'un autre, ne peut pas le déclencher directement. C'est le schéma de connexions qui dicte le déclenchement.

3.3.2.6 Intégration de tâches fréquentes dans l'architecture

Afin de faciliter la tâche au développeur, des tâches fréquentes ont été incorporées à l'architecture logicielle. Par exemple, la librairie dynamique encapsulant le système d'exploitation rend aisée l'utilisation de sémaphores, de communication réseau, etc. L'utilisation de *Player* évite la programmation bas niveau, avec ports série et autre.

Une librairie de communication a également été conçue dans le cadre du projet et a été intégrée à *Acropolis*. Cette librairie permet de sérialiser des structures de données, de les envoyer via le réseau TCP/IP à une autre plate-forme ou un autre programme sur la même plate-forme et de reconstruire les structures de données. Cette communication ne nécessite aucunement la spécification d'adresse IP : chaque programme se connecte en donnant son nom, et les programmes s'échangent des informations grâce à ce nom. Enfin, la gestion des connexions et déconnexions est incluse : chaque fois qu'une nouvelle plate-forme se connecte ou se déconnecte, toutes les autres plates-formes en sont notifiées.

3.3.2.7 Intégration de la visualisation dans l'architecture

Malheureusement, le temps a manqué afin de créer une interface graphique standardisée de visualisation pour *Acropolis*. Plusieurs modules sont capables de fournir des données visualisables. Une seule console ne peut suffire à afficher de façon ordonnée toutes ces informations.

Une interface graphique de visualisation permettrait à chaque module d'*Acropolis* de communiquer avec un module de visualisation de l'interface. Il serait possible d'activer ou de désactiver la visualisation de certains modules. Finalement, des commandes pourraient être envoyées aux modules d'*Acropolis* via l'interface.

3.3.2.8 Possibilité de simuler le fonctionnement de la plate-forme

L'utilisation de *Player* simplifie grandement la simulation : les pilotes *Stage* supportent déjà cette fonctionnalité pour plusieurs capteurs et actuateurs.

La simulation est complètement transparente pour *Acropolis* : rien ne doit être changé. Pour *Player*, il suffit de lui fournir un fichier de configuration différent lui spécifiant de charger les pilotes de simulation ainsi qu'une carte de l'environnement virtuel. Aucune recompilation n'est nécessaire.

3.3.2.9 Evolutivité

Tous les arguments précédents devraient suffire pour affirmer qu'*Acropolis* est une architecture évolutive. Comme illustré antérieurement à la figure 3.5, il est possible d'échanger, d'intercaler et de synchroniser des modules. Des modules génériques peuvent être conçus. L'intégration de deux schémas de connexions ainsi que la

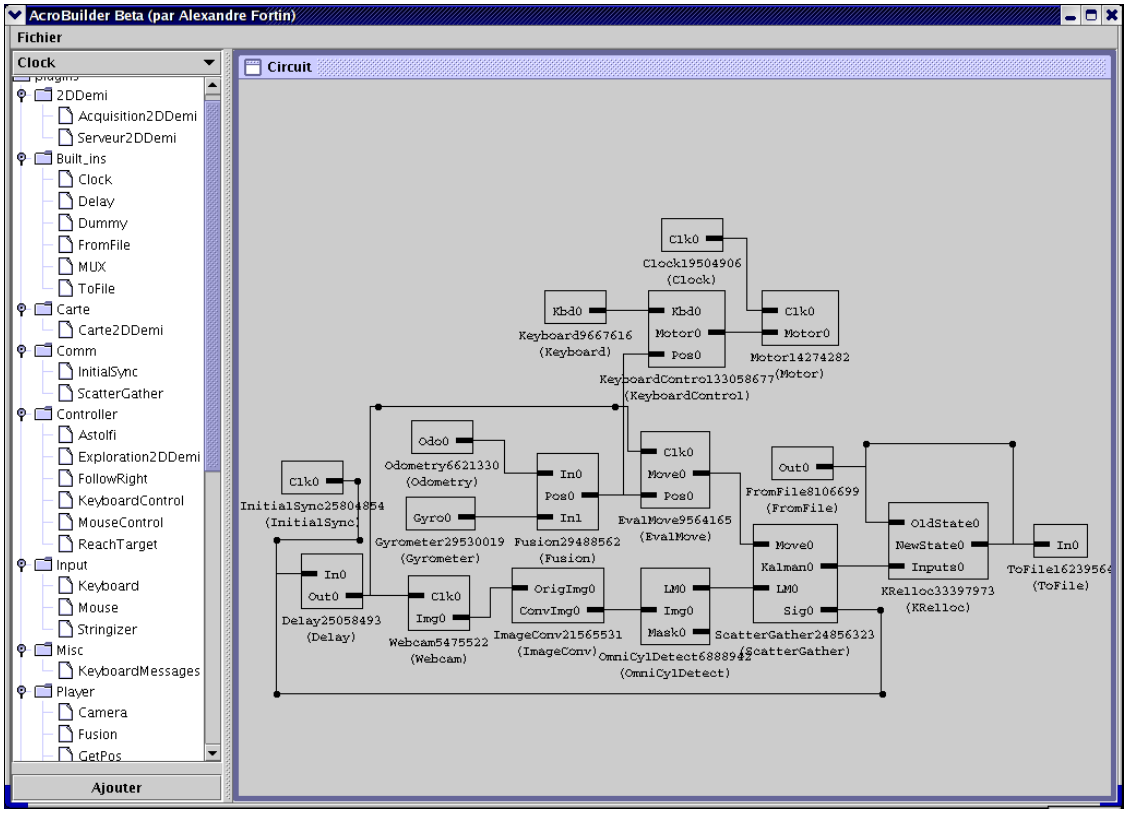


FIG. 3.6 L'interface graphique *AcroBuilder*, lors de la création d'un fichier de configuration XML pour *Acropolis*

conception d'un nouveau module ne requiert aucune recompilation.

3.3.3 Interface graphique *AcroBuilder*

Grâce au travail de M. Alexandre Fortin, étudiant en maîtrise au GRPR, une interface graphique a été conçue afin de faciliter la création de schémas de connexions pour *Acropolis*. Les figures montrant des schémas de connexions dans ce chapitre ont été créées grâce à *AcroBuilder*. L'interface dans son ensemble est montrée à la figure 3.6.

Ainsi, pour utiliser *Acropolis*, il faut au préalable créer un fichier de configuration

en XML décrivant le schéma de connexions. Plutôt que d'écrire ce fichier manuellement, utiliser une interface graphique telle que *AcroBuilder* est beaucoup plus simple, rapide, efficace et agréable.

3.4 Conclusion

Une architecture logicielle permettant l'abstraction matérielle et algorithmique a été développée dans le cadre de ce projet. Son utilisation permet une grande flexibilité dans le développement d'algorithmes en robotique mobile et accroît l'efficacité des développeurs. Elle utilise *Player*comme outil d'abstraction matérielle, mais propose néanmoins quelques innovations au niveau de l'abstraction algorithmique, comme le mécanisme de déclenchement externe des algorithmes, par exemple.

CHAPITRE 4

SYSTÈME DE VISION

Ce chapitre présente le problème de vision résolu afin de fournir des informations supplémentaires pour corriger la localisation relative mutuelle des plates-formes mobiles. Le choix de la caméra omnidirectionnelle sera justifié et comparé avec d'autres types de senseurs, puis une brève présentation de la théorie projective de cette caméra sera fournie. L'algorithme de vision développé sera ensuite détaillé, en indiquant ses forces et ses faiblesses.

4.1 Problème de vision

Le problème de vision rencontré dans le projet comporte deux grandes questions :

- Comment détecter un point de repère ?
- Comment mettre en correspondance un même point de repère détecté par plusieurs plates-formes ?

Le choix de la méthode de détection des points de repère dépend de plusieurs facteurs :

- le type de senseur visuel utilisé,
- l'environnement (structuré, semi-structuré, non structuré),
- l'éclairage de l'environnement (artificiel et contrôlé, naturel, taille des ombres)
- les caractéristiques des points de repère (couleur, forme, texture),

Le problème de correspondance est influencé par les points suivants :

- Les points de repère sont-ils connus d'avance ?
- Est-il possible d'avoir deux points de repère semblables ?

– D'autres facteurs peuvent-ils aider à la mise en correspondance (entourage immédiat des points de repère, contraintes géométriques de positionnement des points de repère) ?

Le projet de localisation relative mutuelle et celui du développement d'une architecture logicielle évolutive de prototypage rapide étant suffisamment complexes et lourds pour un projet de maîtrise, nous avons simplifié le problème de vision posé. La simplification introduite a tout de même l'avantage de permettre l'illustration complète des autres éléments et la preuve de concept de toute notre approche. La flexibilité de l'architecture logicielle permettra à de futurs chercheurs de remplacer le module de vision développé pour ce projet par un module plus flexible et plus robuste, sans avoir de problèmes d'intégration avec l'implantation du système de localisation relative mutuelle.

Ainsi, afin de faciliter le travail de détection, des points de repère connus d'avance ont été utilisés. De plus, chacun d'entre eux est unique et donc facilement identifiable. L'éclairage artificiel contrôlé permet également de simplifier grandement l'algorithme de détection.

Cependant, afin de garder une certaine généralité, aucune supposition n'a été émise au sujet de la connaissance *a priori* de la position des points de repère ni de l'aspect de l'environnement, outre l'éclairage.

4.2 Choix du senseur

Il est nécessaire pour chaque plate-forme de détecter le plus de points de repère possible simultanément afin de permettre le plus de correspondances possible entre les plates-formes. Plus il y a redondance dans l'information visuelle, plus la correction apportée à la localisation sera précise.

À cette fin, un grand champ visuel est nécessaire. Les systèmes de vision permettant un tel champ sont généralement dans l'une des quatre catégories présentées à la figure 4.1[1] : les caméras rotatives, les caméras à oeil-de-poisson[2], les systèmes catadioptriques et les essaims de caméras.

4.2.1 Caméras rotatives

La première idée qui vient à l'esprit afin de détecter des objets sur 360° est d'utiliser une caméra sur une base pivotante (Doncarli et al., 1991). Ainsi, une caméra usuelle munie d'une lentille photographique normale peut être utilisée.

Les inconvénients d'un tel système sont multiples. Premièrement, une mécanique supplémentaire est nécessaire afin de tourner la caméra, ce qui demande conception, installation et énergie supplémentaire. De plus, un système d'encodeur doit être utilisé afin de connaître l'orientation de la caméra. Un problème de synchronisation survient : il est nécessaire de récupérer les données d'encodeur au moment précis où la caméra obtient une image. Des erreurs d'angle plus grandes que pour les autres solutions sont encourues avec cette méthode. La rotation induit également un flou de mouvement dans l'image, ce qui peut nuire à son analyse. Enfin, il est impossible d'avoir toutes les détections simultanément, ce qui complique grandement la fusion ultérieure des données.

---

[1] Avec la permission de Dan Slater pour la figure 4.1(b) et la permission de *Point Grey Research Inc* pour la figure 4.1(d)

[2] Traduction de l'anglais « fish-eye ».

(a) Caméra rotative

(b) Caméra à oeil-de-poisson

(c) Système catadioptrique

(d) Essaim de caméras

FIG. 4.1 Catégories de systèmes de vision à grand angle d'ouverture

4.2.2 Caméras à oeil-de-poisson

Une lentille oeil-de-poisson peut être fixée sur une caméra afin de changer son angle d'ouverture, permettant d'obtenir un champ de 180°. Aucune mécanique de rotation n'est nécessaire et la simultanéité des détections est assurée. Il est possible d'obtenir un champ de 360° en utilisant deux de ces caméras dos-à-dos, ou, si seulement un demi-espace vertical est suffisant, en plaçant la caméra verticalement (Micusik and Pajdla, 2003).

Il est à noter que les objets en périphérie d'une telle lentille subissent un fort effet de perspective, rendant difficile la détection de points de repère. L'inconvénient de ce système est la nécessité d'une lentille spéciale, qui peut s'avérer passablement chère. Outre le prix de la lentille, il faut être capable de fixer la lentille sur la caméra, ce qui élimine l'utilisation d'une webcam et suggère l'utilisation d'une caméra analogique accompagnée d'un saisisseur d'image[3] afin d'en numériser la sortie. Le montage complet s'avère plus coûteux que la solution suivante.

4.2.3 Systèmes catadioptriques

Un système catadioptrique, également appelé caméra omnidirectionnelle, est un senseur visuel associant à une caméra une surface réfléchissante afin d'augmenter son champ de vision. De telles surfaces permettent d'obtenir un angle d'ouverture dépassant même 180°, ce qui peut être très utile pour obtenir une vue panoramique d'une scène (Barth and Ishiguro, 1994; Dellaert and Stroupe, 2002; Kato et al., 1999; Spletzer et al., 2001).

Comme dans le cas d'une lentille oeil-de-poisson, un système catadioptrique peut

[3]Traduction de l'anglais « frame grabber ».

simultanément détecter des objets tout autour d'une plate-forme en plaçant la caméra verticalement. Cependant, en raison du champ supérieur à 180°, l'effet de perspective est moins prononcé que dans le cas d'une caméra oeil-de-poisson. Toutefois, la surface réfléchissante induit une déformation non linéaire de l'image projetée de la scène, ce qui complique la localisation des objets détectés.

Un système catadioptrique ne nécessite pas (dans la plupart des cas) de lentille spécialisée : l'utilisation d'une webcam devient possible, ce qui élimine le besoin d'un saisississeur d'image. Le fait qu'un miroir courbe de qualité est souvent moins cher qu'une lentille oeil-de-poisson de qualité équivalente suggère que le coût d'un système catadioptrique de qualité est souvent moindre que celui d'une caméra à oeil-de-poisson.

Le principal problème du système catadioptrique réside dans la faible précision de l'image obtenue. En effet, une caméra qui, dans le cas d'une *QuickCam Pro 4000*, a un champ de vision de 43° × 33°, couvre 3,35 % de l'espace avec 307 200 pixels, alors qu'un système catadiotrique tel que celui conçu dans le cadre du projet a un champ de vision de 360° × 106°, couvrant 63,8 % de l'espace pour le même nombre de pixels. Finalement, afin d'obtenir une projection adéquate, il est important de bien calibrer la position de la caméra par rapport à la surface réfléchissante, ce qui induit des erreurs supplémentaires.

4.2.4 Essaims de caméras

Une quatrième méthode permettant d'accroître le champ visuel est d'utiliser plusieurs caméras ayant des orientations différentes simultanément, puis de fusionner les images obtenues afin de construire une image panoramique (Ikeda et al., 2003). Cette solution donne des images beaucoup plus précises que les précédentes, tout

en conservant la simultanéité des détections.

Malheureusement, cette solution a plusieurs problèmes d'ordre technologiques. Premièrement, les caméras utilisées doivent être munies d'un mécanisme de synchronisation afin que la prise d'images soit simultanée. Ensuite, il faut utiliser un algorithme performant de fusion d'images afin de générer l'image panoramique résultante. Si un taux d'acquisition d'images élevé est nécessaire, la quantité d'informations devant transiter dans l'unité de traitement devient rapidement colossale ; un ordinateur spécialisé devient alors nécessaire. Tout cela laisse présager un coût élevé.

4.2.5 Senseur retenu

Le système catadioptrique a été choisi comment étant le meilleur compromis. Son faible coût et la simplicité de conception matérielle et logicielle de la solution sont les facteurs déterminants de cette décision. Il est à noter que le système donnant les meilleurs résultats est l'essaim de caméras, mais sa complexité d'implantation dépassait l'envergure du projet. La section 4.3 décrit avec plus de détails la théorie du système catadioptrique utilisé.

La figure 4.2 présente un schéma du système utilisé pour détecter les points de repère autour des plates-formes mobiles (une photographie du système se trouve à la figure 2.4). Le tableau 4.1 donne les paramètres de la caméra et de la surface réfléchissante utilisées donnés par les fabricants. La distance focale de la webcam a été mesuré empiriquement grâce au « Camera Calibration Toolbox » de *Matlab*. Le paramètre c a été calculé grâce à l'équation (4.2).

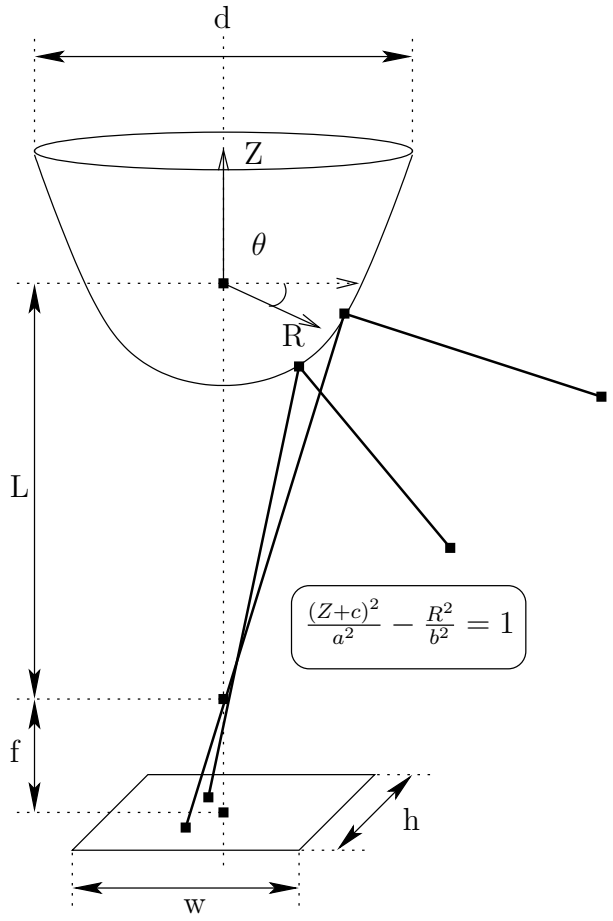


FIG. 4.2 Système catadioptrique utilisé

TAB. 4.1 Paramètres du système catadioptrique

Caméra	QuickCam Pro 4000
Fabricant	Logitech
Résolution utilisée	640×480
Fréquence de trames	5 images/sec
Distance focale f	807 pixels
Miroir	Hyperboloïdal
Fabricant	Neovision
Paramètre a	28.0950 mm
Paramètre b	23.4125 mm
Paramètre c	36.5715 mm
Diamètre d	60.0000 mm
Distance $L = 2c$	73.1430 mm

4.3 Résumé de la théorie projective de la caméra omnidirectionnelle

Cette section donne un aperçu de la théorie des caméras omnidirectionnelles. D'abord, la problématique du centre de projection effectif unique est abordée, puis les équations de projection et de projection inverse d'un tel système sont présentées.

4.3.1 Centre de projection effectif unique

En théorie projective, le centre de projection réel d'une caméra munie d'une lentille en perspective normale est aussi appelé le trou d'épingle. Il s'agit du point de l'espace par où passent tous les rayons émanant de la scène avant d'être projetés sur le plan image.

Un système catadioptrique a un centre de projection réel, le trou d'épingle de la caméra utilisée. En effet, tous les rayons réfléchis sur le plan image passent par ce point. Cependant, sous certaines conditions, le système peut également avoir un centre de projection effectif (voir figure 4.3). Si, dans le cas où la surface réfléchissante n'existait pas, tous les rayons émanant de la scène passaient par un même point, alors ce point pourrait être considéré comme le centre de projection d'un capteur virtuel (Baker and Nayar, 1999).

L'avantage de concevoir un système catadioptrique ayant un centre de projection effectif unique est de pouvoir obtenir, à partir de l'image omnidirectionnelle projetée, l'image en projection perspective qu'aurait obtenu une caméra usuelle si elle avait été placée au centre de projection effectif. Une image panoramique (en projection cylindrique) peut également être obtenue à partir de la même image omnidirectionnelle (Nayar, 1997).

Pour obtenir un centre de projection effectif unique, un système catadioptrique doit

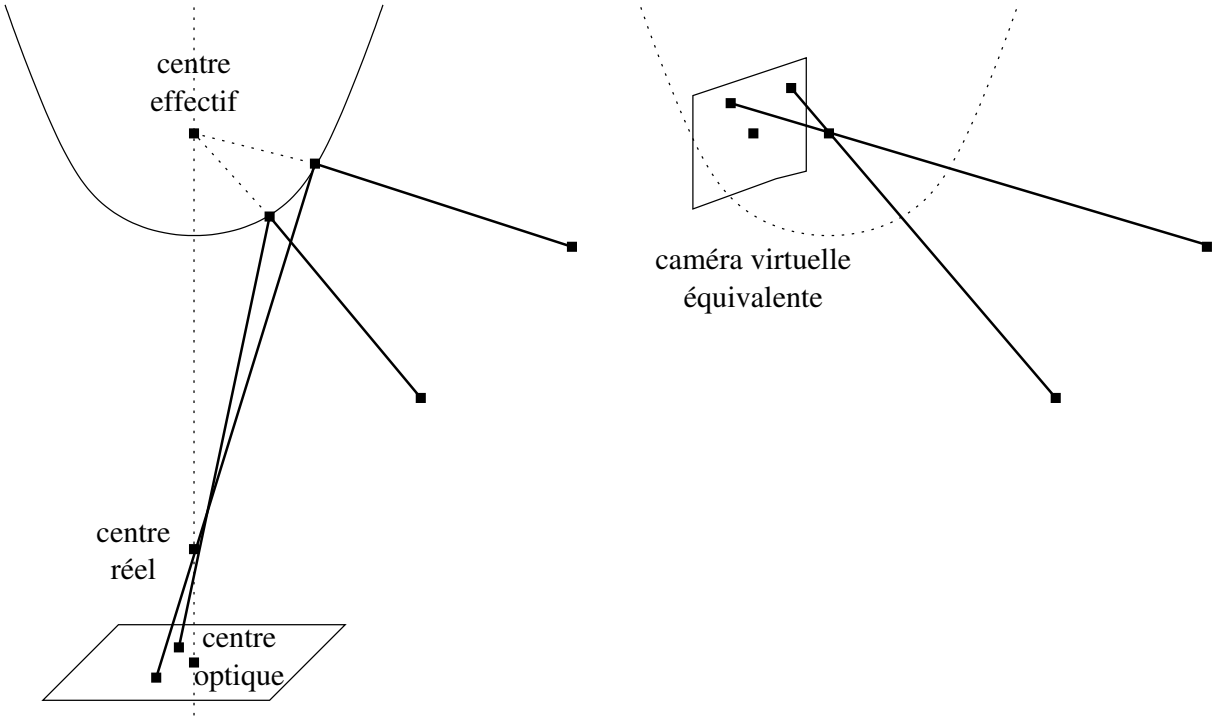


FIG. 4.3 Centre de projection effectif d'un système catadioptrique

être conçu avec l'une des six surfaces réfléchissantes suivantes (Baker and Nayar, 1999) :

Miroir plan : peu utile, car il n'augmente pas le champ visuel.

Miroir conique : solution dégénérée ; le centre de projection effectif doit être à l'apex du cône (tous les rayons doivent toucher la pointe du cône).

Miroir sphérique : solution dégénérée ; le centre de projection effectif ainsi que le centre de projection réel doivent être au centre de la sphère, ce qui fait que seuls les points compris à l'intérieur du miroir sphérique sont visibles.

Miroir ellipsoïdal : solution adéquate, mais peu utilisée dans la pratique car le champ visuel est limité à $360^{\circ} \times 90^{\circ}$, contrairement au cas suivant.

Miroir hyperboloïdal : une des deux solutions les plus utilisées ; aucune lentille spéciale n'est nécessaire, mais le centre de projection réel doit absolument être situé au deuxième foyer de l'hyperboloïde (Yamazawa et al., 1993).

Miroir paraboloïdal : solution la plus utilisée ; calibrage facile, car le centre de projection réel peut être situé n'importe où sur l'axe de la paraboloïde (Nayar, 1997). Cette solution nécessite toutefois l'utilisation d'une lentille télécentrique pour obtenir une projection orthographique.

Comme l'utilisation d'une webcam simplifie beaucoup l'implantation et réduit les coûts, nous avons optés pour le miroir hyperboloïdal, car il est impossible d'utiliser de lentille télécentrique en conjonction avec une webcam.

4.3.2 Propriétés de l'hyperboloïde

L'hyperboloïde est une surface de révolution générée à partir d'une hyperbole. Cela permet de remplacer les coordonnées cartésiennes usuelles par un système de coordonnées cylindriques (voir figure 4.2). L'avantage d'un tel système d'axes est de pouvoir ensuite travailler en deux dimensions, dans un plan vertical $\theta = \theta^*$, sans perte de généralité. Il est à noter que le centre du système d'axes est au centre de projection effectif du système catadioptrique, qui est également un des deux foyers de l'hyperboloïde.

L'équation de l'hyperboloïde, en coordonnées cylindriques, centrée au foyer, est donnée par :

$$\frac{(Z+c)^2}{a^2} - \frac{R^2}{b^2} = 1 \tag{4.1}$$

où a est la distance entre le centre de l'hyperboloïde et l'apex, c est la distance entre le centre et le foyer (voir figure 4.4) et où b satisfait l'équation :

$$a^2 + b^2 = c^2 \tag{4.2}$$

Une propriété importante de l'hyperboloïde (et de l'hyperbole) stipule qu'une droite

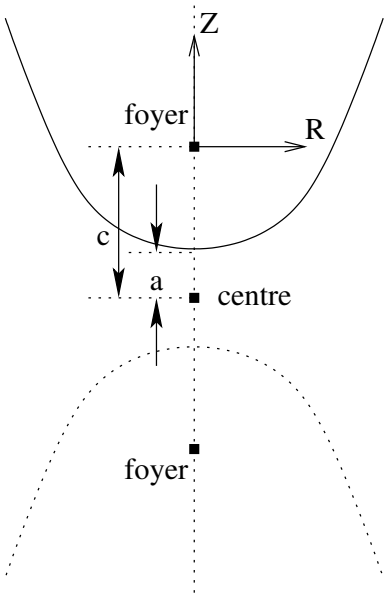


FIG. 4.4 Définition de l'hyperboloïde

passant par un des foyers de l'hyperboloïde est réfléchi en une droite passant par le second foyer de l'hyperboloïde (voir figure 4.5). C'est cette propriété qui permet d'obtenir un centre de projection effectif unique : le foyer de l'hyperboloïde.

Enfin, une autre propriété intéressante de l'hyperboloïde montre que la projection d'une droite parallèle à l'axe de l'hyperboloïde (une droite verticale, dans le cas qui nous intéresse) donne une droite passant par le centre optique dans le plan image. Ainsi, tout objet de la scène ayant une arête verticale devrait générer une arête radiale dans l'image omnidirectionnelle, ce qui peut être utile pour détecter des points de repère.

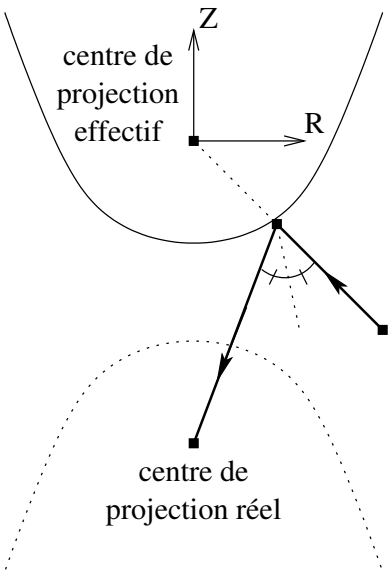


FIG. 4.5 Propriété de réflexion de l'hyperbole et de l'hyperboloïde

4.3.3 Équation de projection

Un point (R, θ, Z) de la scène est projeté en un point (r, θ) du plan image selon la relation :

$$r = \frac{q_1 f R}{-q_2 Z + q_3 \sqrt{R^2 + Z^2}} \tag{4.3}$$

où :

$$q_1 = c^2 - a^2, \quad q_2 = c^2 + a^2, \quad q_3 = 2ac \tag{4.4}$$

Ainsi, l'angle θ est le même dans la scène et dans le plan image. Le point $(0, 0)$ du plan image correspond au centre optique. r et f sont exprimés dans les mêmes unités, c'est-à-dire en pixels. Pour les détails de calculs, se référer à l'annexe I.

4.3.4 Équation de projection inverse

En divisant par Z les numérateur et dénominateur de l'équation (4.3), on obtient :

$$r = \frac{q_1 f \frac{R}{Z}}{-q_2 + \text{signe}(Z) q_3 \sqrt{\left(\frac{R}{Z}\right)^2 + 1}} \tag{4.5}$$

Il devient possible d'isoler $\frac{R}{Z}$:

$$\frac{R}{Z} = \frac{q_1 q_2 f + q_3 \sqrt{r^2 \left(q_2^2 - q_3^2\right) + \left(q_1 f\right)^2}}{\left|\left(q_1 f\right)^2 - \left(q_3 r\right)^2\right|} \cdot r \tag{4.6}$$

Cette équation est la droite à laquelle appartient le point de scène (R, θ, Z) qui s'est projeté au pixel (r, θ) du plan image. Il est impossible avec ces seules informations de retrouver le point de scène.

Il est toutefois possible de poser certaines hypothèses permettant de calculer la projection inverse. La plus simple est de supposer $Z = Z_0$ connu. Dans ce cas, R peut être calculé avec :

$$R = \frac{q_1 q_2 f + q_3 \sqrt{r^2 \left(q_2^2 - q_3^2\right) + \left(q_1 f\right)^2}}{\left|\left(q_1 f\right)^2 - \left(q_3 r\right)^2\right|} \cdot r Z_0 \tag{4.7}$$

4.4 Choix des points de repère

Comme dit précédemment, la problème de vision a été réduit afin de ne pas déborder du cadre de ce projet de maîtrise. À cette fin, des balises simples à détecter et à identifier (voir figure 4.6) ont été conçues.

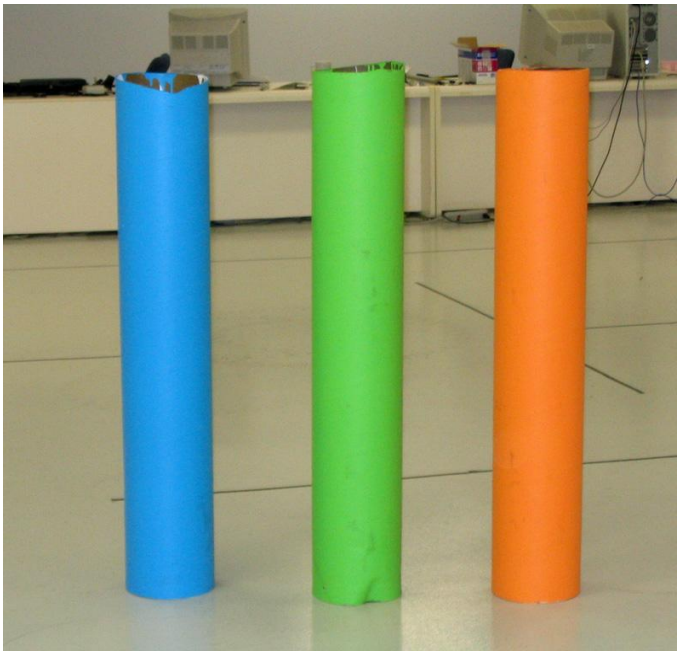


FIG. 4.6 Balises conçues dans le cadre du projet pour servir de points de repère

Les balises sont de couleurs uniformes afin d'en faciliter la détection. En premier lieu, il s'agit, pour détecter une balise, de trouver une plage[4] de pixels ayant à peu près la même couleur.

Les balises sont de forme cylindrique et toujours à la verticale afin d'éliminer les mauvaises détections. Ainsi, une plage uniforme détectée sera considérée comme un point de repère si elle est filiforme et si le vecteur principal de la plage est dirigé vers le centre de l'image omnidirectionnelle. Comme mentionné à la sous-section 4.3.2, les lignes verticales de la scène sont projetées en lignes radiales dans l'image omnidirectionnelle.

Enfin, chaque balise a une couleur différente de celle des autres afin de faciliter l'identification du point de repère.

[4]Traduction de l'anglais « blob »

4.5 Algorithmes

Cette section décrit les algorithmes utilisés afin de détecter, identifier, localiser et étalonner les balises cylindriques de la section 4.4 grâce au système catadioptrique présenté à la sous-section 4.2.5.

4.5.1 Détection, localisation et identification des balises

Dans le cadre du projet, les plates-formes mobiles connaissent *a priori* les différentes couleurs possibles pour les balises. Cette hypothèse est fortement contraignante et devra être levée par des développements futurs.

Grâce à cette connaissance, il est possible de chercher une balise à la fois : c'est ce que l'algorithme fait. La détection, la localisation et l'identification de la balise ne nécessitent qu'une seule traversée des pixels de l'image, mais on doit répéter une traversée par couleur connue de balise.

4.5.1.1 Espace de couleur HSV

Avant de commencer la détection, un prétraitement est appliqué à l'image omnidirectionnelle. Il s'agit en fait de changer de l'espace de couleur RGB à l'espace de couleur HSV. Cette étape, contrairement aux autres, n'est effectuée qu'une seule fois pour toutes les balises.

L'espace de couleur HSV a trois dimensions : la teinte (« Hue »), la saturation (« Saturation ») et la valeur (« Value »). La teinte décrit à elle seule la couleur d'un pixel et peut être visualisée comme un cercle où chaque angle correspond à une couleur. La saturation détermine si une couleur est fade (faible saturation) ou

vive (forte saturation). La valeur indique la quantité de lumière : une faible valeur donnera une couleur sombre, alors qu'une forte valeur donnera une couleur bien éclairée.

L'espace HSV est particulièrement bien adapté à la détection de pixels d'une certaine couleur. Alors que l'espace RGB usuel nécessite un intervalle de seuillage dans chacune des trois dimensions, l'espace HSV définit la couleur grâce en grande partie à la teinte, et accessoirement à la saturation. La valeur est souvent influencée davantage par l'éclairage de la scène que par la couleur de la surface perçue. Comme la teinte n'est pratiquement pas influencée par l'éclairage, la détection d'une couleur dans l'espace de couleur HSV est beaucoup plus fiable.

4.5.1.2 Seuillage

En pratique, on ne peut toutefois se fier qu'à la teinte. Lorsque la valeur est très basse, c'est-à-dire dans les zones sombres (les zones d'ombres, notamment), la teinte et la saturation ne sont plus numériquement fiables. Il est alors préférable de choisir des pixels en se basant sur les trois coordonnées.

Ainsi, chaque balise est caractérisée par un intervalle très serrée de teinte et un intervalle beaucoup plus étendu de saturation. Finalement, toutes les balises ont le même intervalle de valeur. Ce troisième intervalle accepte pratiquement toutes les valeurs, sauf les faibles valeurs, qui correspondent aux zones trop sombres pour qu'on puisse y distinguer une couleur avec précision et assurance.

Ainsi, lors de la traversée de l'image HSV pour détecter une balise particulière, la première étape consiste à marquer les pixels qui respectent les trois intervalles de la balise.

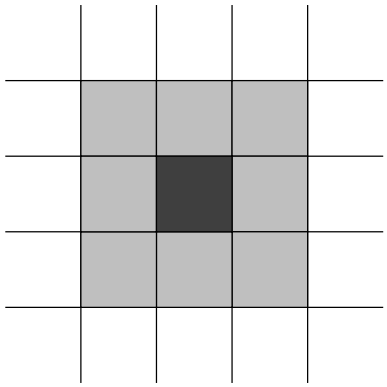


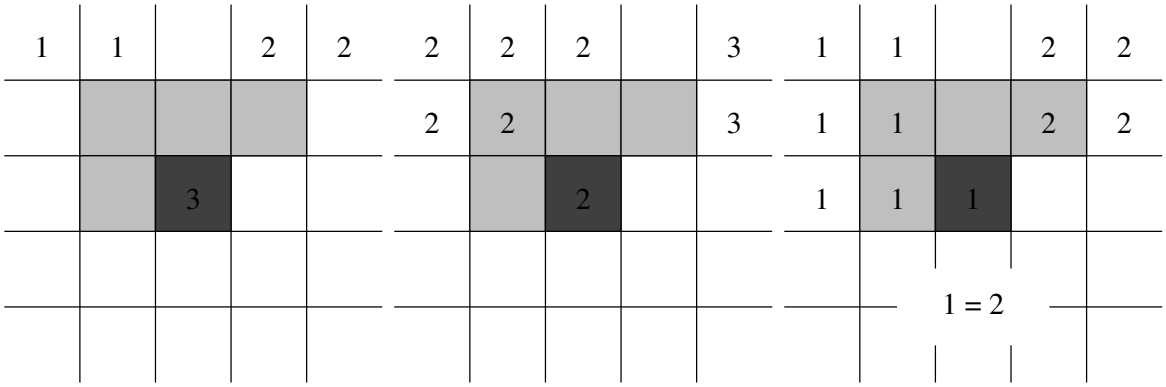
FIG. 4.7 La 8-connexité utilisée lors de la segmentation

4.5.1.3 Segmentation en plages

Afin de déterminer la forme à laquelle un pixel appartient, il faut regrouper tous les pixels marqués connexes en plages : c'est la segmentation. À cette fin, la 8-connexité a été utilisée, comme montré à la figure 4.7. À chaque plage est assignée un identifiant pour analyse future, et tous ses pixels constitutifs sont marqués avec cet identifiant.

La subtilité de l'algorithme consiste à faire le tout en une seule traversée, la même que celle qui effectue le seuillage. Par conséquent, lors de l'analyse d'un pixel, seuls quatre de ses huit voisins ont déjà été analysés, comme le montrent les exemples de la figure 4.8. Après seuillage du pixel, s'il est marqué, un identifiant de plage doit lui être attribué. Trois cas se présentent :

1. Aucun des quatre voisins visités n'est marqué (figure 4.8(a)). Dans ce cas, un nouvel identifiant est attribué au pixel.
2. Certains des voisins visités sont marqués du même identifiant (figure 4.8(b)). Le pixel prend alors lui aussi cet identifiant.
3. Certains des voisins visités sont marqués, mais avec deux identifiants différents (figure 4.8(c)). Dans ce cas, un des identifiants est attribué au pixel et



(a) Nouvelle plage (b) Suite d'une plage (c) Fusion implicite de deux plages

FIG. 4.8 Cas possibles lors de la segmentation

on ajoute à une table de correspondance le couple d'identifiants, signifiant qu'ils représentent tous deux la même plage.

4.5.1.4 Sélection de la meilleure plage

La prochaine étape est de déterminer quelle plage de pixels a le plus de chance d'être la balise de la couleur recherchée, si une telle plage existe. Étant donné que la balise est cylindrique et verticale par hypothèse, et parce que les lignes verticales sont projetées en lignes radiales dans l'image omnidirectionnelle (voir section 4.3.2), l'algorithme simple qui suit peut être appliqué à chaque plage pour en estimer la qualité.

Soit la meilleure droite (au sens des moindres carrés) passant au travers du nuage de pixels d'une plage et obligatoirement par le centre de l'image omnidirectionnelle (puisque les arêtes verticales d'un cylindre seront projetées en lignes radiales). Cette droite donne l'orientation du point de repère dans le système de coordonnées de la plate-forme mobile l'ayant détecté. Si la plage est réellement la projection d'un

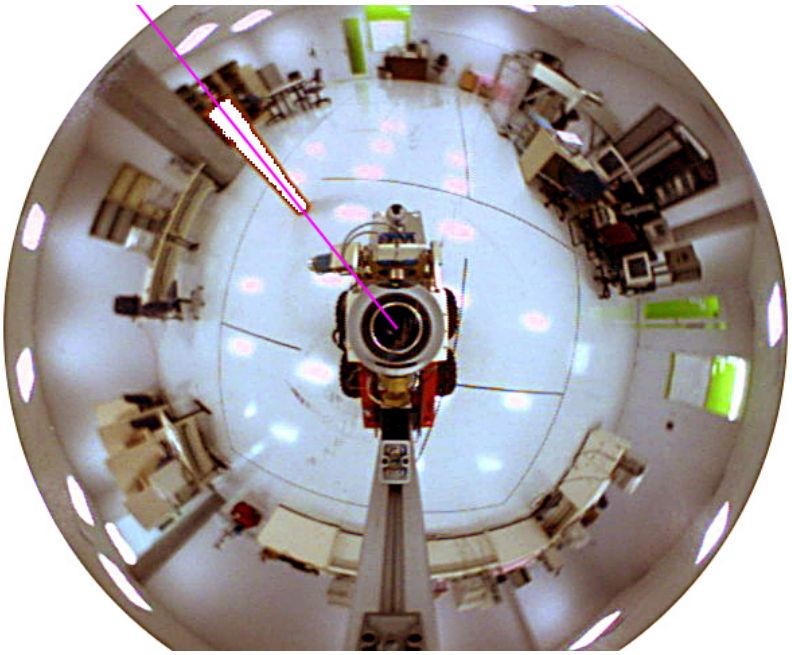


FIG. 4.9 Meilleure droite passant au travers d'une plage correspondant à une balise cylindrique, dans l'image omnidirectionnelle ; les pixels de la plage sont serrés autour de la droite

cylindre vertical, alors les pixels marqués devraient être serrés autour de la droite, comme le montre la figure 4.9.

Du point de vue statistique, la droite minimise la somme des carrés des erreurs, notée S. Si cette somme des carrés des erreurs est faible, alors cela signifie que les points sont en général proches de la droite, laissant présager un objet filiforme, comme la projection d'un cylindre. Si, en plus, l'aire de la plage, c'est-à-dire le nombre de pixels la constituant, est assez élevée, alors la plage peut être considérée comme un point de repère avec un niveau de confiance élevé.

Afin de déterminer laquelle des plages de pixels sera retenue comme point de repère, un critère de qualité Q doit être élaboré. Q devrait être d'autant plus grand que la plage est filiforme (que S est petit) et devrait également être proportionnel au nombre de pixels N d'une plage afin de privilégier les plus grandes plages.

La définition suivante du critère de qualité Q a été utilisée dans le cadre du projet :

$$Q = \frac{N^{2,5}}{S} \tag{4.8}$$

Les détails de calculs à propos de la méthode des moindres carrés utilisée pour calculer S et l'orientation θ de la meilleure droite se trouvent à l'annexe II. Cet algorithme permet de calculer le critère Q sans avoir à faire plus d'une traversée des pixels par point de repère.

Après la segmentation, le critère Q est calculé pour chaque plage. Si le critère Q d'aucune plage ne dépasse le seuil fixé empiriquement, alors la détection est considérée comme ayant échoué. Sinon, la plage ayant le plus grand critère Q est retenue comme point de repère de la couleur analysée.

4.5.1.5 Détermination de l'angle de la plage

L'angle θ obtenu à l'équation (II.14), l'angle de la demi-droite, correspond à l'orientation du point de repère par rapport à la plate-forme mobile l'ayant détecté. Il reste à déterminer la distance à laquelle se trouve le point de repère, si cela est possible.

4.5.1.6 Détermination de la distance de la plage

Comme mentionné à la sous-section 4.3.4, il est impossible de déterminer la distance réelle du point de repère à partir de l'image omnidirectionnelle sans information supplémentaire.

L'hypothèse que le sol de l'environnement est principalement plat a déjà été émise

à la section 2.1. Si l'on suppose également que les points de repère touchent le sol, alors il devient possible d'estimer sans grande précision la position du point de contact du point de repère avec le sol à partir de l'image omnidirectionnelle. Il devient nécessaire de connaître la distance séparant le centre de projection effectif du sol, c'est-à-dire la hauteur du foyer du miroir hyperboloïdal.

Dans ce cas, l'équation (4.7) peut être utilisée pour calculer la distance du point de repère à partir de la distance radiale du pixel. Cette dernière est la distance entre le centre optique et la projection du point de contact du point de repère avec le sol. Ce pixel est facilement identifiable : c'est le pixel de la plage qui est le plus proche du centre optique de l'image omnidirectionnelle.

Afin de déterminer ce pixel efficacement, il suffit d'examiner, pixel par pixel, une droite partant du centre optique et ayant pour angle celui calculé précédemment grâce à l'équation (II.14). Dès qu'un pixel marqué comme appartenant à la plage en question est rencontré, la distance de ce pixel au centre optique peut être calculée.

Pour encore plus d'efficacité, l'algorithme classique de tracé de ligne de *Bresenham* a été implanté pour parcourir la droite dans l'image omnidirectionnelle.

4.5.2 Étalonnage des balises

Afin que la détection des points de repère soit fiable, les intervalles de couleur discutés à la sous-section 4.5.1.2 doivent être déterminés avec soin. De plus, la position de la webcam doit être suffisamment précise pour avoir un centre de projection effectif unique. À cette fin, une procédure de calibrage semi-automatique a été élaborée.

Un patron est superposé à l'image omnidirectionnelle et la webcam est ensuite déplacée verticalement afin de faire correspondre le contour de l'image omnidirec-

tionnelle avec le patron. Puis, les balises cylindriques sont placées à un endroit bien précis de la scène pour que la procédure puisse analyser leurs pixels constitutifs.

Après la conversion de l'image dans l'espace de couleur HSV et la construction d'histogrammes afin d'analyser la distribution des pixels, des intervalles de H, S et V sont déterminés. Pour plus d'informations au sujet de la procédure semi-automatique d'étalonnage des balises, se référer à l'annexe III.

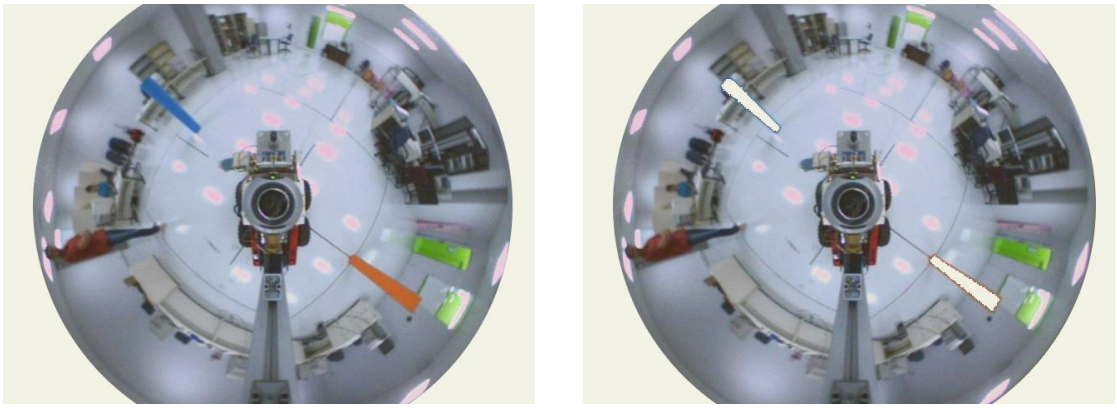
4.6 Discussion de la méthode

Quoique simple, l'algorithme de vision développé dans ce projet est le fruit du travail de l'auteur. Il repose sur des algorithmes connus d'analyse et de traitement d'images et ne contient pas de contribution scientifique à ce niveau.

La méthode de détection développée dans ce chapitre est très contraignante au niveau du choix et de l'emplacement des points de repère. Toutefois, l'algorithme performe très bien dans un environnement contrôlé, atteignant des taux de réussite quasi-parfaits à moins de cinq mètres. La figure 4.10 donne un exemple de résultat obtenu avec l'algorithme de détection de balise cylindrique.

Au-delà de cette distance, l'imprécision de la webcam jumelée à l'effet du miroir pose problème et l'algorithme tend à ne pas détecter les points de repère. Néanmoins, il est excessivement rare qu'un objet imprévu soit détecté comme étant un point de repère.

L'algorithme utilisé pour le seuillage est très sensible à l'éclairage de l'environnement. Si de grandes zones ombragées existent, ou si l'éclairage est variable, la chance de détection diminue rapidement. Par conséquent, l'algorithme est inutilisable tel quel à l'extérieur.



(a) Image omnidirectionnelle originale (b) Résultat de l'algorithme de détection

FIG. 4.10 Application de l'algorithme de détection

Il serait probablement peu compliqué de rendre la détection plus robuste aux variations d'intensité lumineuse. Entre autre, le calibrage des intervalles de couleur est déjà semi-automatique ; il est possible d'automatiser entièrement cette opération. Par exemple, des patrons de couleur pourraient être placés à l'arrière du miroir pour que la caméra puisse périodiquement ajuster ses paramètres afin de percevoir les couleurs de la même façon, peu importe l'éclairage.

Telle que présentée, la méthode a une faille non négligeable : l'algorithme de segmentation repose sur la 8-connexité. Or, si un obstacle, comme une simple barre en métal par exemple, fait obstruction partielle à une balise cylindrique, alors le cylindre sera segmenté en deux plages : une plage d'un côté de la barre et une plage de l'autre côté. Ces deux plages ne seront pas suffisamment filiformes pour être retenues comme points de repère et la détection échouera. Une procédure de fusion de plages rapprochées pourrait être implantée afin de régler ce problème.

CHAPITRE 5

FILTRAGE DE KALMAN

Ce chapitre présente la méthode utilisée pour mettre à jour et corriger l'estimation de la localisation relative mutuelle, c'est-à-dire le filtrage de Kalman.

Dans un premier temps seront démontrées les équations régissant la mise à jour de la localisation, indépendamment de l'utilisation du filtre de Kalman étendu. La notation y sera également présentée (section 5.1). Ensuite, le choix d'une solution utilisant le filtre de Kalman sera justifié (section 5.2). Les équations du filtre lui-même seront ensuite calculées (section 5.3). Enfin, quelques détails et problèmes d'implantation seront exposés (section 5.4).

5.1 Système d'équations et contraintes

La définition de la localisation relative mutuelle a été présentée à la section 2.2.1. Le premier problème consiste à exprimer la localisation actuelle en fonction de la localisation de l'itération précédente et des trajectoires que les différentes plates-formes se sont communiquées. Définissons mathématiquement d'abord la localisation, puis la trajectoire et après sera établi le système d'équations.

5.1.1 Expression de la localisation

Soit N plates-formes mobiles a, b, ..., N[1], effectuant de la localisation relative mutuelle. Exprimons la localisation dans le repère de a à l'itération k de l'algorithme. Comme mentionné précédemment, il est inutile d'ajouter la propre localisation relative de a, puisqu'elle sera $\mathbf{p}_{aa}^k = (0, 0, 0)$ pour tout k. Parce que toutes les plates-formes seront localisées dans le repère de a (sans perte de généralité aucune), nous nous permettons également d'omettre le 2$^{\text{e}}$ indice inférieur indiquant le repère utilisé afin d'alléger les équations.

L'expression de la localisation relative mutuelle est :

$$[x_b^k \quad y_b^k \quad \theta_b^k \quad x_c^k \quad y_c^k \quad \theta_c^k \quad \cdots \quad x_N^k \quad y_N^k \quad \theta_N^k]^{\mathrm{T}} \tag{5.1}$$

5.1.2 Expression de la trajectoire

La technique employée discrétise le temps en quanta afin de synchroniser les différentes plates-formes. Par conséquent, il n'est nécessaire que de connaître le déplacement des plates-formes sans connaître la forme de la trajectoire réelle. En effet, lorsque viendra le temps de calculer la nouvelle pose d'une plate-forme, une opération sera appliquée sur la pose précédente, et cette opération sera indépendante de la forme de la trajectoire.

En d'autres termes, que le robot ait parcouru la distance le séparant de la pose de l'itération $k-1$ à la pose de l'itération k en ligne droite ou en décrivant un arc de cercle n'influence pas les mesures prises dans la pose finale.

[1] Nous nous permettons un abus de langage : bien que N représente un nombre de plates-formes, nous l'utiliserons en indice afin d'indiquer la dernière plate-forme.

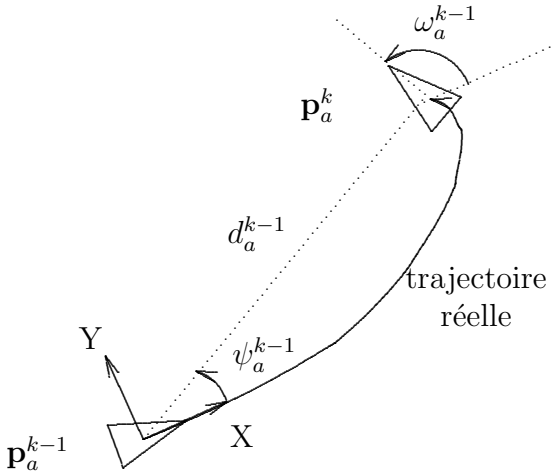


FIG. 5.1 Définition de la trajectoire

Par conséquent, il convient de choisir un mode de représentation simple du déplacement d'une plate-forme. Tout déplacement rigide (n'impliquant aucune déformation de la plate-forme) peut s'exprimer sous la forme d'une translation suivie d'une rotation.

En deux dimensions, la translation comporte deux degrés de liberté. Les choix habituels sont les coordonnées cartésiennes et les coordonnées polaires. Dans le cas actuel, il est préférable de choisir les coordonnées polaires, car elles sont plus fidèles à ce qui se passe dans la réalité.

En effet, lorsque viendra le temps de quantifier l'incertitude sur les déplacements des robots, il est plus naturel de donner une incertitude en translation longitudinale et une incertitude en rotation que des incertitudes en X et Y. De plus, ce choix de coordonnées simplifie les équations subséquentes.

La translation sera donc exprimée comme un couple (d, ψ), c'est-à-dire une distance d parcourue dans la direction ψ par rapport à l'avant de la plate-forme. La rotation sera simplement exprimée sous la forme d'un angle ω. La figure 5.1 illustre ces définitions.

En fonction de ce qui précède, nous pouvons maintenant exprimer l'information disponible à la plate-forme a à l'itération k grâce à l'intercommunication, c'est-à-dire les trajectoires effectuées à partir de chaque pose $k-1$ pour se rendre à chaque pose k. Il est à noter que, contrairement à l'expression de la localisation dans le repère de a uniquement, chaque trajectoire est exprimée ici dans le repère de la plate-forme l'ayant effectuée.

$$[d_a^{k-1} \quad \psi_a^{k-1} \quad \omega_a^{k-1} \quad d_b^{k-1} \quad \psi_b^{k-1} \quad \omega_b^{k-1} \quad \cdots \quad d_N^{k-1} \quad \psi_N^{k-1} \quad \omega_N^{k-1}]^{\mathrm{T}} \tag{5.2}$$

5.1.3 Système d'équations

À partir de la localisation (5.1) de a à l'itération $k-1$ et des trajectoires (5.2) obtenues par communication à l'itération k, il faut maintenant calculer la nouvelle localisation à l'itération k. Il faut donc appliquer les déplacements à toutes les plates-formes, puis appliquer un changement de repère pour ramener celui-ci au nouveau centre de a.

5.1.3.1 Déplacement de a

La première étape est d'exprimer les poses de toutes les plates-formes dans l'ancien repère de a. L'ancienne pose de a dans ce repère, soit $(0, 0, 0)$, simplifie les calculs. Le cas particulier du déplacement de a sera observé avant le cas général.

Il faut d'abord appliquer une translation de longueur d_a^{k-1} dans la direction ψ_a^{k-1} par rapport à l'orientation précédente de a, soit 0. Il s'agit d'additionner ensuite

ω_a^{k-1} à l'orientation précédente pour obtenir la nouvelle pose :

$$\check{\mathbf{p}}_a^k = \begin{bmatrix} \check{x}_a^k \\ \check{y}_a^k \\ \check{\theta}_a^k \end{bmatrix} = \begin{bmatrix} d_a^{k-1}\cos(\psi_a^{k-1}) \\ d_a^{k-1}\sin(\psi_a^{k-1}) \\ \omega_a^{k-1} \end{bmatrix} \tag{5.3}$$

La notation $(\check{\cdot})$ dénote le fait que ces valeurs sont encore exprimées dans l'ancien repère de a.

5.1.3.2 Déplacement des autres plates-formes

Il faut maintenant appliquer les mêmes opérations aux autres plates-formes. Pour une plate-forme $i \neq a$, la translation $(d_i^{k-1}, \psi_i^{k-1})$ et la rotation ω_i^{k-1} sont exprimées dans le repère de i. Il faut donc effectuer un changement de repère. De plus, la pose initiale n'est plus $(0,0,0)$, mais plutôt $(x_i^{k-1}, y_i^{k-1}, \theta_i^{k-1})$. La figure 5.2 illustre le problème.

Comme on peut le constater, la translation $(d_i^{k-1}, \psi_i^{k-1})$, une fois exprimée dans le repère de a, devient $(d_i^{k-1}, \theta_i^{k-1} + \psi_i^{k-1})$, alors que la rotation demeure ω_i^{k-1} dans les deux repères. La nouvelle pose de i dans l'ancien repère de a devient :

$$\check{\mathbf{p}}_i^k = \begin{bmatrix} \check{x}_i^k \\ \check{y}_i^k \\ \check{\theta}_i^k \end{bmatrix} = \begin{bmatrix} x_i^{k-1} + d_i^{k-1}\cos(\theta_i^{k-1} + \psi_i^{k-1}) \\ y_i^{k-1} + d_i^{k-1}\sin(\theta_i^{k-1} + \psi_i^{k-1}) \\ \theta_i^{k-1} + \omega_i^{k-1} \end{bmatrix} \tag{5.4}$$

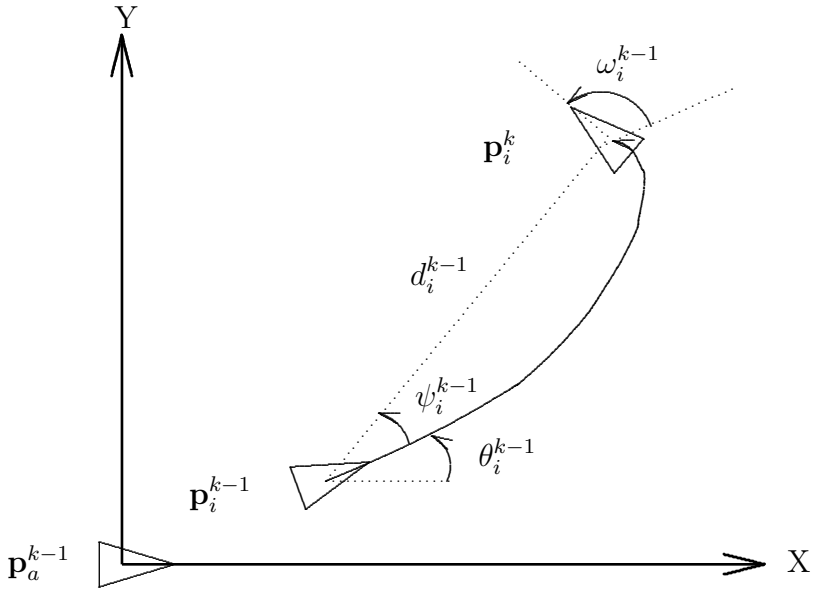


FIG. 5.2 Déplacement d'une plate-forme i dans le repère de a

5.1.3.3 Changement de repère

Toutes les poses ont maintenant été exprimées dans l'ancien repère de a. Toutefois, la contrainte imposée par la définition de localisation relative mutuelle à la sous-section 5.1.1 stipule que la pose de a doit toujours être $(0, 0, 0)$. Par conséquent, il faut trouver un changement de repère qui amène la pose $(\check{x}_a^k, \check{y}_a^k, \check{\theta}_a^k)$ à la pose $(0, 0, 0)$ et l'appliquer à toutes les plates-formes : il s'agit d'une translation de $(-\check{x}_a^k, -\check{y}_a^k)$, suivie d'une rotation de $-\check{\theta}_a^k$. Sera tout d'abord calculée la nouvelle position d'une plate-forme i, suivie de son orientation.

Afin de calculer la position de i, les deux transformations sont appliquées :

$$\begin{aligned}\begin{bmatrix} x_i^k \\ y_i^k \end{bmatrix} &= \begin{bmatrix} \cos(-\check{\theta}_a^k) & -\sin(-\check{\theta}_a^k) \\ \sin(-\check{\theta}_a^k) & \cos(-\check{\theta}_a^k) \end{bmatrix} \left(\begin{bmatrix} \check{x}_i^k \\ \check{y}_i^k \end{bmatrix} + \begin{bmatrix} -\check{x}_a^k \\ -\check{y}_a^k \end{bmatrix} \right) \\ &= \begin{bmatrix} \cos(\check{\theta}_a^k) & \sin(\check{\theta}_a^k) \\ -\sin(\check{\theta}_a^k) & \cos(\check{\theta}_a^k) \end{bmatrix} \begin{bmatrix} \check{x}_i^k - \check{x}_a^k \\ \check{y}_i^k - \check{y}_a^k \end{bmatrix}\end{aligned} \tag{5.5}$$

Après substitution des équations (5.3) et (5.4) dans l'équation (5.5) :

$$\begin{bmatrix} x_i^k \\ y_i^k \end{bmatrix} = \begin{bmatrix} \cos(\omega_a^{k-1}) & \sin(\omega_a^{k-1}) \\ -\sin(\omega_a^{k-1}) & \cos(\omega_a^{k-1}) \end{bmatrix} \begin{bmatrix} x_i^{k-1} + d_i^{k-1}\cos(\theta_i^{k-1} + \psi_i^{k-1}) - d_a^{k-1}\cos(\psi_a^{k-1}) \\ y_i^{k-1} + d_i^{k-1}\sin(\theta_i^{k-1} + \psi_i^{k-1}) - d_a^{k-1}\sin(\psi_a^{k-1}) \end{bmatrix} \tag{5.6}$$

Pour ce qui est de l'orientation de i, la translation n'a aucune influence. Seule la rotation du repère de a change l'orientation relative de i. Après rotation et substitution de l'équation (5.4) :

$$\begin{aligned} \theta_i^k &= \check{\theta}_i^k - \omega_a^{k-1} \\ &= \theta_i^{k-1} + \omega_i^{k-1} - \omega_a^{k-1} \end{aligned} \tag{5.7}$$

5.1.4 Expression de la mesure

Les équations obtenues jusqu'à maintenant permettent d'évaluer la localisation relative mutuelle en boucle ouverte, c'est-à-dire sans correction des erreurs qui s'accumulent au fil du temps. Afin de réduire ces erreurs, il faut intégrer une rétroaction grâce aux mesures de position de points de repères. Comme ils ne se déplacent pas, il est utile d'estimer leurs positions. Elles serviront à confirmer la bonne localisation des plates-formes et à fermer la boucle de contrôle.

Les points de repère se verront attribués les indices numériques 1, 2, ... La position du point de repère ℓ exprimée dans le repère de la plate-forme i à l'itération k est représentée par le couple $\mathbf{p}_{\ell i}^k = (x_{\ell i}^k, y_{\ell i}^k)$.

Ces positions seront estimées à partir des mesures prises avec la caméra omnidirectionnelle montée sur chaque plate-forme. Après le traitement et l'analyse d'image décrits au chapitre 4, il est possible d'obtenir la position du point de repère ℓ à

partir des mesures $(r_{\ell i}^k, \phi_{\ell i}^k)$ prises par la plate-forme i, où $r_{\ell i}^k$ est le rayon (en pixels) du centre de l'image de la caméra omnidirectionnelle de la plate-forme i jusqu'au plus proche pixel appartenant au point de repère ℓ. Ce pixel correspond au point de plus faible altitude du point de repère. $\phi_{\ell i}^k$ est l'angle que fait ce même pixel avec le centre de l'image. Il s'agit donc de coordonnées polaires centrées sur le pixel correspondant au centre du miroir hyperboloïdal dans l'image, avec $\phi_{\ell i}^k = 0$ correspondant à l'axe longitudinal de la plate-forme.

Il est nécessaire de rappeler qu'il est possible de déterminer la direction d'un point de repère avec une seule image omnidirectionnelle, mais qu'il n'est pas possible d'en déterminer la distance sans hypothèse supplémentaire. Voici les deux plus simples :

1. Supposer que la hauteur du point de repère est connue, ce qui permet de la mettre en correspondance avec la hauteur du point de repère en pixel dans l'image. À l'aide de triangles semblables, on obtient la distance du point de repère.
2. Supposer que le sol est plat et que les points de repère touchent au sol. Dans ce cas, il est possible de faire correspondre le point de contact du point de repère avec le sol et le pixel du point de repère le plus proche du centre de l'image. Par application du théorème de Pythagore, on obtient la distance du point de repère.

Dans le cadre du projet, lorsqu'une mesure de distance est utilisée, la seconde hypothèse a été choisie.

Grâce à la modélisation de la prise de mesure effectuée précédemment dans le chapitre 4, il est possible de prévoir la mesure à partir de la localisation relative mutuelle. La plate-forme a peut estimer la mesure du point de repère ℓ prise par

la plate-forme i à l'itération k ainsi[2] :

$$r_{\ell i}^k = \frac{q_1 f \sqrt{\left(x_\ell^k - x_i^k\right)^2 + \left(y_\ell^k - y_i^k\right)^2}}{q_2 Z_i + q_3 \sqrt{\left(x_\ell^k - x_i^k\right)^2 + \left(y_\ell^k - y_i^k\right)^2 + Z_i^2}} \tag{5.8}$$

$$\phi_{\ell i}^k = \operatorname{atan2}\left(y_\ell^k - y_i^k, x_\ell^k - x_i^k\right) - \theta_i^k \tag{5.9}$$

La distance Z_i est la hauteur du foyer du miroir par rapport au sol sur la plate-forme i, f est la distance focale de la webcam et les paramètres q_1, q_2 et q_3 dépendent du miroir hyperboloïdal.

5.1.5 Rétroaction

La seule information pouvant être utilisée pour corriger la localisation relative mutuelle des plates-formes est la mesure de la position des points de repère.

Si la plate-forme a possède un estimé de la position du point de repère ℓ dans son repère à l'itération $k-1$, alors elle peut prédire la position de ℓ à l'itération k dans son nouveau repère puisque, par hypothèse, les points de repère ne se déplacent pas. L'équation (5.6) exprimant la position d'une plate-forme dans le nouveau repère de a peut donc être utilisée pour donner la position du point de repère, en posant que son déplacement est nul :

$$\begin{bmatrix} x_\ell^k \\ y_\ell^k \end{bmatrix} = \begin{bmatrix} \cos(\omega_a^{k-1}) & \sin(\omega_a^{k-1}) \\ -\sin(\omega_a^{k-1}) & \cos(\omega_a^{k-1}) \end{bmatrix} \begin{bmatrix} x_\ell^{k-1} - d_a^{k-1}\cos(\psi_a^{k-1}) \\ y_\ell^{k-1} - d_a^{k-1}\sin(\psi_a^{k-1}) \end{bmatrix} \tag{5.10}$$

[2]Nous utilisons $\operatorname{atan2}(y, x)$ pour dénoter la fonction `C atan2(double y, double x)`. Il s'agit de la fonction mathématique $\arctan\frac{y}{x}$, mais qui tient compte du quadrant dans lequel se trouve le point (x, y). L'image de la fonction $\operatorname{atan2}(y, x)$ est $[-\pi, \pi]$.

Puisque a peut prédire la position de ℓ, les équations (5.8) et (5.9) peuvent être utilisées pour prédire la position du pixel de ℓ dans l'image de n'importe quelle plate-forme.

Par conséquent, la contrainte principale du système est que la prédiction, telle que calculée par a, des positions de ℓ dans les images de toutes les plates-formes l'ayant détecté doit être le plus proche possible des mesures qu'ont effectuées les plates-formes réellement. Il est alors envisageable de trouver une méthode optimale pour rajuster la localisation relative mutuelle afin de satisfaire cette contrainte.

5.2 Choix de la méthode

Les deux méthodes considérées dans le casdre de ce projet, soit la méthode de minimisation des moindres carrées de Levenberg-Marquardt (More, 1977) et le filtre de Kalman étendu (Bierman, 1977), seront présentées ici, avec une discussion de leurs avantages et inconvénients respectifs.

5.2.1 Méthode de Levenberg-Marquardt

La localisation relative mutuelle des plates-formes et des points de repère à l'itération $k-1$ est disponible à la plate-forme a, ainsi que les trajectoires $(d_i^{k-1}, \psi_i^{k-1}, \omega_i^{k-1})$ des plates-formes. Comme développé précédemment, il est maintenant possible de calculer une estimation *a priori*, c'est-à-dire avant intégration des mesures, de la localisation relative mutuelle à l'itération k grâce aux équations (5.6) et (5.7).

La plate-forme a possède également les mesures de vision de toutes les plates-formes. Grâce aux équations (5.8) et (5.9), il est possible d'obtenir une estimation *a priori* de ces mesures qui est fonction de la qualité de l'estimation *a priori* de

la localisation. S'il y a un grand écart entre mesures estimées et mesures réelles, alors soit les mesures réelles sont très bruitées, soit l'erreur de l'estimation est importante.

Par conséquent, une méthode d'optimisation classique peut être utilisée pour trouver une estimation *a posteriori* de la localisation qui soit proche de l'estimation *a priori*, tout en minimisant l'écart entre les mesures réelles et estimées, en pondérant les équations en fonction de la confiance attribuée à la localisation *a priori* et aux mesures réelles.

Pour procéder plus avant, deux notations supplémentaires sont nécessaires : $(\tilde{\cdot})$ dénote une estimation *a priori* et $(\hat{\cdot})$, une estimation *a posteriori*. De plus, M^k sera le nombre de points de repère détectés par l'ensemble des plates-formes à l'itération k. Finalement, une variable est requise pour indiquer si la plate-forme i a détecté le point de repère ℓ à l'itération k :

$$D_{\ell i}^k = \begin{cases} 1 & \text{si } i \text{ a détecté } \ell \text{ à l'itération } k, \\ 0 & \text{sinon.} \end{cases}$$

Un exemple simplifié de fonction à minimiser, où les poids ont été omis, pourrait ressembler à :

$$\sum_{i=a}^{N} \left[\left(\tilde{x}_i^k - \hat{x}_i^k \right)^2 + \left(\tilde{y}_i^k - \hat{y}_i^k \right)^2 + \left(\tilde{\theta}_i^k \ominus \hat{\theta}_i^k \right)^2 + \sum_{\ell=1}^{M^k} D_{\ell i}^k \cdot \left[\left(\tilde{r}_{\ell i}^k - \hat{r}_{\ell i}^k \right)^2 + \left(\tilde{\phi}_{\ell i}^k \ominus \hat{\phi}_{\ell i}^k \right)^2 \right] \right] \quad (5.11)$$

L'opérateur $\ominus$ est défini comme étant le plus petit angle entre deux orientations. Il sert à éviter les effets nocifs qu'ont les discontinuités de représentation angulaire

sur la convergence de l'algorithme. L'opérateur a pour propriété : $\alpha \ominus \beta \in (-\pi, \pi]$.

Après substitution des valeurs estimées *a priori*, on remarque que l'équation est non linéaire. Une méthode fréquemment employée pour les problèmes de minimisation de moindres carrés est l'algorithme de Levenberg-Marquardt. Il s'agit d'une méthode itérative qui converge vers la solution optimale, si celle-ci est unique.

L'avantage principal de cette méthode est qu'elle procure une solution optimale si elle converge, même dans les cas non linéaires (contrairement à la prochaine solution).

L'inconvénient principal est la lenteur de l'algorithme. À chaque itération k, une sous-procédure elle-même itérative devra être appelée, ce qui ne présage rien de bon pour les applications en temps réel. De plus, la situation peut être catastrophique en cas de non convergence ou de minimum local.

5.2.2 Filtre de Kalman étendu

À la lumière des développements de la section précédente, la seconde méthode envisagée est le filtre de Kalman étendu.

Le filtre de Kalman est une version récursive du cas linéaire de minimisation des moindres carrés. Il donne la solution optimale à un problème d'optimisation, en utilisant les calculs effectués aux itérations précédentes. Par conséquent, le calcul à chaque iteration k est simple et rapide à effectuer, car il utilise récursivement les résultats de l'itération $k-1$ sans nécessiter une sous-boucle de convergence elle-même itérative.

Cependant, le filtre ne s'applique que dans le cas linéaire. Néanmoins, une version du filtre, nommée filtre de Kalman étendu, permet d'utiliser des équations simi-

laires pour les systèmes non linéaires. Toutefois, la solution ainsi obtenue n'est plus nécessairement optimale au sens des moindres carrés. Il n'en demeure pas moins qu'en pratique, ce filtre est abondamment utilisé en robotique mobile car il procure efficacement de bons résultats (Welch and Bishop, 2003).

Le lien récursif avec l'itération précédente tient dans la construction et la mise à jour d'une matrice de covariance de l'erreur d'estimation. De plus, grâce à la définition de matrices de covariance du bruit de processus et du bruit de mesure, le filtre permet de gérer la confiance attribuée à l'estimation et aux mesures.

Finalement, il est également possible d'utiliser le filtre de Kalman étendu pour estimer des états passés, présents et futurs (Welch and Bishop, 2003), ce qui peut s'avérer très utile en robotique mobile.

5.3 Équations non linéaires du filtre de Kalman

Une courte définition, se basant sur une introduction aux filtres de Kalman (Welch and Bishop, 2003), sera maintenant présentée, suivie de l'élaboration du système d'équations du projet.

5.3.1 Brève définition du filtre de Kalman étendu

Le problème posé est d'estimer un état $\mathbf{x}^k \in \mathbb{R}^n$ (il y a donc n variables d'état), qui varie selon l'équation stochastique :

$$\mathbf{x}^k = \mathbf{f}\left(\mathbf{x}^{k-1}, \mathbf{u}^{k-1}, \mathbf{w}^{k-1}\right) \tag{5.12}$$

La fonction vectorielle $\mathbf{f}$ est le modèle de processus (à concevoir). Le vecteur $\mathbf{u}^{k-1} \in$

$\mathbb{R}^p$ est la commande (connue) envoyée au système. Le vecteur $\mathbf{w}^{k-1} \in \mathbb{R}^q$ est le bruit de processus (inconnu), supposé blanc, gaussien et de moyenne nulle.

Afin de corriger l'estimation, on dispose du vecteur de mesure $\mathbf{z}^k \in \mathbb{R}^m$ (il y a donc m mesures), lui-même issu du modèle non linéaire suivant :

$$\mathbf{z}^k = \mathbf{h}\left(\mathbf{x}^k, \mathbf{v}^k\right) \tag{5.13}$$

La fonction vectorielle $\mathbf{h}$ est le modèle de mesure (à concevoir). Le vecteur $\mathbf{v}^{k-1} \in \mathbb{R}^t$ est le bruit de mesure (inconnu), supposé blanc, gaussien et de moyenne nulle.

Après avoir modélisé le système avec ces deux fonctions, il est possible d'estimer *a priori* l'état actuel et les mesures en utilisant l'état corrigé (ou *a posteriori*) précédent et en supposant les bruits nuls :

$$\tilde{\mathbf{x}}^k = \mathbf{f}\left(\hat{\mathbf{x}}^{k-1}, \mathbf{u}^{k-1}, \mathbf{0}\right) \tag{5.14}$$

$$\tilde{\mathbf{z}}^k = \mathbf{h}\left(\tilde{\mathbf{x}}^k, \mathbf{0}\right) \tag{5.15}$$

Une nouvelle matrice peut maintenant être définie : la matrice $n \times n$ de la covariance de l'erreur d'estimation, $\mathbf{P}^k$. C'est grâce à cette matrice que le filtre de Kalman attribue plus ou moins de confiance à l'estimation de l'état, une covariance de l'erreur faible signifiant que l'estimation est fiable. Cette matrice vaut :

$$\mathbf{P}^k = E\left[\left(\mathbf{x}^k - \hat{\mathbf{x}}^k\right)\left(\mathbf{x}^k - \hat{\mathbf{x}}^k\right)^{\mathrm{T}}\right] \tag{5.16}$$

Une estimation *a priori* de cette matrice peut être définie à partir de l'estimation *a priori* de l'état :

$$\tilde{\mathbf{P}}^k = E\left[\left(\mathbf{x}^k - \tilde{\mathbf{x}}^k\right)\left(\mathbf{x}^k - \tilde{\mathbf{x}}^k\right)^{\mathrm{T}}\right] \tag{5.17}$$

À partir de ces définitions, le filtre calcule une matrice $n \times m$ de gain $\mathbf{K}^k$ qui permet de pondérer la différence entre mesures réelles et estimées afin de créer une correction à apporter à l'état. Cette correction est optimale au sens des moindres carrés dans le cas où $\mathbf{f}$ et $\mathbf{h}$ sont linéaires.

La version non linéaire du filtre effectue donc une linéarisation du système, ce qui nécessite la définition de quelques matrices jacobiennes (l'exposant k signifiant toujours le numéro de l'itération) :

$$\begin{aligned} \mathbf{A}^k &= \left[a_{ij}^k\right] && \text{une matrice } n \times n \\ \mathbf{W}^k &= \left[w_{ij}^k\right] && \text{une matrice } n \times q \\ \mathbf{H}^k &= \left[h_{ij}^k\right] && \text{une matrice } m \times n \\ \mathbf{V}^k &= \left[v_{ij}^k\right] && \text{une matrice } m \times t \end{aligned}$$

où :

$$a_{ij}^k = \frac{\partial f_i}{\partial x_j}\left(\hat{\mathbf{x}}^{k-1}, \mathbf{u}^{k-1}, \mathbf{0}\right) \tag{5.18}$$

$$w_{ij}^k = \frac{\partial f_i}{\partial w_j}\left(\hat{\mathbf{x}}^{k-1}, \mathbf{u}^{k-1}, \mathbf{0}\right) \tag{5.19}$$

$$h_{ij}^k = \frac{\partial h_i}{\partial x_j}\left(\tilde{\mathbf{x}}^k, \mathbf{0}\right) \tag{5.20}$$

$$v_{ij}^k = \frac{\partial h_i}{\partial v_j}\left(\tilde{\mathbf{x}}^k, \mathbf{0}\right) \tag{5.21}$$

Ceci permet d'écrire les fonctions linéarisées :

$$\mathbf{x}^k \approx \tilde{\mathbf{x}}^k + \mathbf{A}^k\left(\mathbf{x}^{k-1} - \hat{\mathbf{x}}^{k-1}\right) + \mathbf{W}^k\mathbf{w}^{k-1} \tag{5.22}$$

$$\mathbf{z}^k \approx \tilde{\mathbf{z}}^k + \mathbf{H}^k\left(\mathbf{x}^k - \tilde{\mathbf{x}}^k\right) + \mathbf{V}^k\mathbf{v}^k \tag{5.23}$$

Il reste finalement à définir des matrices de covariance de bruit de processus et de bruit de mesure :

$$\mathbf{Q}^k = E\left[\mathbf{w}^k \left(\mathbf{w}^k\right)^{\mathrm{T}}\right] \qquad \text{une matrice } q \times q \tag{5.24}$$

$$\mathbf{R}^k = E\left[\mathbf{v}^k \left(\mathbf{v}^k\right)^{\mathrm{T}}\right] \qquad \text{une matrice } t \times t \tag{5.25}$$

Tout est en place pour énoncer les deux étapes du filtre de Kalman étendu. Tout d'abord, la phase de prédiction :

$$\tilde{\mathbf{x}}^k = \mathbf{f}\left(\hat{\mathbf{x}}^{k-1}, \mathbf{u}^{k-1}, \mathbf{0}\right) \tag{5.26}$$

$$\tilde{\mathbf{P}}^k = \mathbf{A}^k \mathbf{P}^{k-1} \left(\mathbf{A}^k\right)^{\mathrm{T}} + \mathbf{W}^k \mathbf{Q}^{k-1} \left(\mathbf{W}^k\right)^{\mathrm{T}} \tag{5.27}$$

puis la phase de correction :

$$\mathbf{K}^k = \tilde{\mathbf{P}}^k \left(\mathbf{H}^k\right)^{\mathrm{T}} \left(\mathbf{H}^k \tilde{\mathbf{P}}^k \left(\mathbf{H}^k\right)^{\mathrm{T}} + \mathbf{V}^k \mathbf{R}^k \left(\mathbf{V}^k\right)^{\mathrm{T}}\right)^{-1} \tag{5.28}$$

$$\hat{\mathbf{x}}^k = \tilde{\mathbf{x}}^k + \mathbf{K}^k \left(\mathbf{z}^k - \mathbf{h}\left(\tilde{\mathbf{x}}^k, \mathbf{0}\right)\right) \tag{5.29}$$

$$\mathbf{P}^k = \left(\mathbf{I} - \mathbf{K}^k \mathbf{H}^k\right) \tilde{\mathbf{P}}^k \tag{5.30}$$

Grâce à ces équations, il est possible d'estimer un état futur ou, à une itération donnée, d'estimer l'état actuel et corriger l'estimation en intégrant les mesures prises, ce qui était recherché.

5.3.2 Système d'équations

Maintenant que tous les outils théoriques sont disponibles, cette section présente les variables et les modèles du système correspondant au problème de localisation

relative mutuelle.

Dans les équations qui suivent, le système sera exprimé en fonction de la plate-forme a pour simplifier la notation, sans aucune perte de généralité. De plus, par hypothèse, le nombre de plates-formes N est fixe et connu. Cependant, nous supposerons pour le moment que le nombre de points de repère est également fixe et connu, soit $M^{k+1} = M^k = M \quad \forall k$. Cette hypothèse supplémentaire sert à simplifier la présentation des équations, mais sera levée à la section 5.4.

5.3.2.1 État

Le vecteur d'état du système doit contenir les paramètres devant être estimés. Pour le problème de localisation, il s'agit de la pose des plates-formes. Toutefois, ce n'est pas suffisant. Afin de se positionner par rapport aux points de repères, il est également nécessaire de tenir à jour l'estimation de leurs positions relatives. Cela donne :

$$\mathbf{x}^k = \begin{bmatrix} x_b^k & y_b^k & \theta_b^k & \cdots & x_N^k & y_N^k & \theta_N^k & x_1^k & y_1^k & \cdots & x_M^k & y_M^k \end{bmatrix}^{\mathrm{T}} \tag{5.31}$$

Ce vecteur est donc de taille $3(N-1) + 2M$.

5.3.2.2 Commande

Le vecteur de commande permet de représenter les paramètres contrôlables du système. C'est dans ce vecteur que l'on définit les variables qui influencent l'état et qui sont fournies au système à chaque itération. Pour la localisation, chaque plate-forme contrôle sa propre trajectoire. Ainsi :

$$\mathbf{u}^k = \begin{bmatrix} d_a^k & \psi_a^k & \omega_a^k & \cdots & d_N^k & \psi_N^k & \omega_N^k \end{bmatrix}^{\mathrm{T}} \tag{5.32}$$

Ce vecteur est donc de taille $3N$.

5.3.2.3 Bruit de processus

La commande envoyée au système peut subir des perturbations avant d'être appliquée. Il est possible que le modèle du processus ne tienne pas compte de certains facteurs inconnus. L'estimation initiale fournie au filtre n'est assurément pas exacte. Pour toutes ces raisons, des variables aléatoires sont ajoutées au modèle du processus afin de permettre au modèle de s'adapter, même à des situations qui n'étaient pas prévues à l'origine.

Comme dans notre cas, le processus dépend fortement de la commande, c'est-à-dire des trajectoires imposées aux plates-formes, une variable de bruit a été définie pour chaque variable de commande, comme suit :

$\bar{d}_i^k$: Erreur de longueur de déplacement
$\bar{\psi}_i^k$: Erreur de direction de déplacement
$\bar{\omega}_i^k$: Erreur d'angle de rotation

Comme il a pu être constaté à l'équation précédente, la notation $\bar{(\cdot)}$ sera utilisée pour représenter le bruit de $(\cdot)$.

Pour permettre plus de flexibilité, une incertitude a également été définie sur la position des points de repère afin de découpler les erreurs de localisation des points de repères aux erreurs dues à l'imprécision du suivi de trajectoire :

$\bar{x}_\ell^k$: Erreur de position en X
$\bar{y}_\ell^k$: Erreur de position en Y

On obtient donc le vecteur de bruit de processus suivant :

$$\mathbf{w}^k = \begin{bmatrix} \bar{d}_a^k & \bar{\psi}_a^k & \bar{\omega}_a^k & \cdots & \bar{d}_N^k & \bar{\psi}_N^k & \bar{\omega}_N^k & \bar{x}_1^k & \bar{y}_1^k & \cdots & \bar{x}_M^k & \bar{y}_M^k \end{bmatrix}^{\mathrm{T}} \tag{5.33}$$

Le vecteur est de taille $3N + 2M$.

5.3.2.4 Modèle du processus

Le modèle du processus est la fonction vectorielle $\mathbf{f}$ de l'équation (5.12), qui permet d'évaluer l'état actuel du système en fonction de l'état précédent, de la commande envoyée à ce moment et du bruit de processus. Il s'agit donc des fonctions permettant de décrire l'évolution de la localisation relative mutuelle après changements de repère, telles que développées aux équations (5.6), (5.7) et (5.10). Voici le système d'équations, en incluant cette fois-ci les bruits de processus :

$$\begin{bmatrix} x_i^k \\ y_i^k \end{bmatrix} = \mathbf{R}^{k-1} \left(\begin{bmatrix} x_i^{k-1} + (d_i^{k-1} + \bar{d}_i^{k-1}) \cos(\theta_i^{k-1} + \psi_i^{k-1} + \bar{\psi}_i^{k-1}) \\ y_i^{k-1} + (d_i^{k-1} + \bar{d}_i^{k-1}) \sin(\theta_i^{k-1} + \psi_i^{k-1} + \bar{\psi}_i^{k-1}) \end{bmatrix} - \mathbf{t}^{k-1} \right) \tag{5.34}$$

$$\theta_i^k = \theta_i^{k-1} + \omega_i^{k-1} + \bar{\omega}_i^{k-1} - (\omega_a^{k-1} + \bar{\omega}_a^{k-1}) \tag{5.35}$$

$$\begin{bmatrix} x_\ell^k \\ y_\ell^k \end{bmatrix} = \mathbf{R}^{k-1} \left(\begin{bmatrix} x_\ell^{k-1} \\ y_\ell^{k-1} \end{bmatrix} - \mathbf{t}^{k-1} \right) + \begin{bmatrix} \bar{x}_\ell^{k-1} \\ \bar{y}_\ell^{k-1} \end{bmatrix} \tag{5.36}$$

avec :

$$\mathbf{R}^{k-1} = \begin{bmatrix} \cos(\omega_a^{k-1} + \bar{\omega}_a^{k-1}) & \sin(\omega_a^{k-1} + \bar{\omega}_a^{k-1}) \\ -\sin(\omega_a^{k-1} + \bar{\omega}_a^{k-1}) & \cos(\omega_a^{k-1} + \bar{\omega}_a^{k-1}) \end{bmatrix} \tag{5.37}$$

$$\mathbf{t}^{k-1} = \begin{bmatrix} (d_a^{k-1} + \bar{d}_a^{k-1}) \cos(\psi_a^{k-1} + \bar{\psi}_a^{k-1}) \\ (d_a^{k-1} + \bar{d}_a^{k-1}) \sin(\psi_a^{k-1} + \bar{\psi}_a^{k-1}) \end{bmatrix} \tag{5.38}$$

5.3.2.5 Mesure

Le vecteur de mesure contient les valeurs lues à partir d'appareils sur les plates-formes (en l'occurence, une caméra omnidirectionnelle). C'est grâce à ces valeurs qu'une correction sera possible.

$$\mathbf{z}^k = \begin{bmatrix} r_{1a}^k & \phi_{1a}^k & \cdots & r_{1N}^k & \phi_{1N}^k & r_{2a}^k & \phi_{2a}^k & \cdots & r_{2N}^k & \phi_{2N}^k & \cdots & r_{MN}^k & \phi_{MN}^k \end{bmatrix}^{\mathrm{T}} \tag{5.39}$$

Ce vecteur est donc de taille $2MN$, pour l'instant. En pratique, chaque robot peut ne pas détecter tous les points de repère.

Il est à noter qu'il pourrait ne pas être désirable de faire l'hypothèse selon laquelle le sol est plat et les points de repère sont en contact avec le sol. Dans ce cas, les mesures de distance ne peuvent être obtenues par l'utilisation d'une seule caméra omnidirectionnelle, il faut recourir à la stéréographie (avec deux caméras omnidirectionnelles, par exemple). Dans ce cas, le vecteur de mesure ne contiendra que les mesures d'angles et aura une taille MN.

5.3.2.6 Bruit de mesure

Les mesures prises par des appareils sont toujours entachées d'erreurs dues aux imprécisions. Il est également possible que le modèle de mesure ne tienne pas compte de certains facteurs inconnus. Par conséquent, des variables aléatoires sont ajoutées au modèle de mesure afin de pondérer en conséquence l'importance des mesures lors de la correction de l'état.

Comme l'incertitude sur la position d'un pixel peut facilement être exprimée et évaluée en termes de la mesure prise de ce pixel, le vecteur de bruit de mesure suis

de près le vecteur de mesure :

$$\mathbf{v}^k = \left[\bar{r}_{1a}^k \;\; \bar{\phi}_{1a}^k \;\; \cdots \;\; \bar{r}_{1N}^k \;\; \bar{\phi}_{1N}^k \;\; \bar{r}_{2a}^k \;\; \bar{\phi}_{2a}^k \;\; \cdots \;\; \bar{r}_{2N}^k \;\; \bar{\phi}_{2N}^k \;\; \cdots \;\; \bar{r}_{MN}^k \;\; \bar{\phi}_{MN}^k\right]^{\mathrm{T}} \tag{5.40}$$

Ce vecteur est donc de taille $2MN$, pour l'instant. En pratique, il a la même taille que le vecteur de mesure, ce qui peut vouloir dire MN si seulement les angles sont retenus.

5.3.2.7 Modèle de mesure

Le modèle de mesure est la fonction vectorielle **h** de l'équation (5.13), qui permet d'évaluer les mesures qui auraient été obtenues si le modèle était parfait et si l'estimation de l'état était exacte. En pratique, cette évaluation sera comparée avec les mesures réelles. La différence sera multipliée par le gain de Kalman et servira de correction. À partir des équations (5.8) et (5.9) et en ajoutant le bruit de mesure, on obtient :

$$r_{\ell i}^k = \frac{q_1 f \sqrt{\left(x_\ell^k - x_i^k\right)^2 + \left(y_\ell^k - y_i^k\right)^2}}{q_2 Z_i + q_3 \sqrt{\left(x_\ell^k - x_i^k\right)^2 + \left(y_\ell^k - y_i^k\right)^2 + Z_i^2}} + \bar{r}_{\ell i}^k \tag{5.41}$$

$$\phi_{\ell i}^k = \operatorname{atan2}\left(y_\ell^k - y_i^k, x_\ell^k - x_i^k\right) - \theta_i^k + \bar{\phi}_{\ell i}^k \tag{5.42}$$

Dans le cas où seules les mesures d'angles sont considérées, l'équation (5.41) n'est plus nécessaire.

5.3.2.8 Autres matrices

Les matrices $\mathbf{A}^k$, $\mathbf{W}^k$, $\mathbf{H}^k$ et $\mathbf{V}^k$ sont des matrices jacobiennes des fonctions vectorielles $\mathbf{f}$ et $\mathbf{h}$. Les détails mécaniques de dérivation des expressions ne sont pas présentés, car ils n'apportent rien de nouveau sur le sujet.

Les matrices $\mathbf{Q}^k$ et $\mathbf{R}^k$ sont des matrices de covariance du bruit de processus et de la mesure, respectivement. Les valeurs de ces matrices se rattachent fortement à l'incertitude que l'on désire affecter aux mesures des appareils utilisés. La diagonale de ces matrices contient en fait les variances des variables aléatoires $\mathbf{w}^k$ et $\mathbf{v}^k$, alors que les autres éléments sont des covariances entre les variables aléatoires.

Les variances peuvent parfois être obtenues à partir de la documentation des appareils utilisés pour le bruit de mesure. Cela s'avère plus difficile pour le bruit de processus, qui tend souvent à modéliser les facteurs inconnus du modèle lui-même. On utilise alors l'expérimentation pour déterminer la variance des variables de bruit.

Si l'on suppose que tous les bruits sont indépendants les uns des autres, alors les matrices $\mathbf{Q}^k$ et $\mathbf{R}^k$ seront diagonales (aucune covariance entre les variables). On dit alors que le bruit est parfaitement non corrélé. Toutefois, cela n'est guère représentatif de la réalité. Malheureusement, il est souvent très difficile de mesurer ces paramètres avec rigueur. Par conséquent, une technique souvent employée pour estimer la covariance entre deux variables aléatoires consiste à utiliser une fraction du produit des écarts-types des deux variables impliquées.

Dans le cadre du projet, les écarts-types des différentes variables de bruit ont été évalués empiriquement. Le carré de ces valeurs a été utilisé pour variances, remplissant les diagonales de $\mathbf{Q}^k$ et $\mathbf{R}^k$. Les 10% du produit des écarts-types ont été utilisés pour la covariance, lorsque les variables pouvaient être corrélées.

La faiblesse apparente de la méthode d'évaluation des matrices $\mathbf{Q}^k$ et $\mathbf{R}^k$ est connue de l'auteur. Une méthode systématique devra être conçue dans le futur pour pallier ce point faible.

5.4 Détails et problèmes d'implantation

Cette section expose plusieurs problèmes rencontrés lors de l'implantation et comment ces problèmes ont été réglés.

5.4.1 Initialisation du filtre de Kalman

De par sa nature récursive, le filtre de Kalman nécessite une estimation initiale de l'état $\mathbf{x}^0$ et de la covariance de l'erreur d'estimation $\mathbf{P}^0$. Souvent, il n'est pas nécessaire de réellement se préoccuper de l'initialisation d'un filtre de Kalman : un état nul avec une covariance suffisamment élevée permet au filtre d'éventuellement converger. Toutefois, dans notre cas, peu de mesures sont disponibles pour corriger l'état, surtout dans le cas où seules les mesures d'angles sont considérées.

Il est donc nécessaire de donner une grossière approximation de la localisation relative mutuelle des plates-formes afin d'initialiser le filtre dans la bonne direction. Le temps ayant manqué au projet, il a été impossible de développer une méthode de localisation approximative non récursive. Par conséquent, l'hypothèse de la section 2.1 stipulant que les plates-formes connaissent approximativement la localisation relative mutuelle initiale est temporairement justifiée. En pratique, une localisation relative mutuelle initiale bruitée a été fournie aux plates-formes.

5.4.2 Nombre de points de repère détectés

À la sous-section 5.3.2, le nombre de points de repère détectés à chaque itération a été supposé fixe et connu à l'avance. De la même manière, l'hypothèse que toutes les plates-formes détectent tous les points de repère à chaque itération fut émise. Tel n'est pas le cas en réalité.

En effet, le premier problème a été de considérer les $D_{\ell i}^k$, c'est-à-dire les variables indiquant si la plate-forme i a détecté le point de repère ℓ à l'itération k. Cela implique que le nombre de mesures n'est pas fixe. À cette fin, les dimensions m et t sont devenues variables, ce qui implique que les dimensions de $\mathbf{z}^k$, $\mathbf{v}^k$, $\mathbf{H}^k$, $\mathbf{V}^k$ et $\mathbf{h}$ sont maintenant variables. Cela en soi est une difficulté plutôt informatique, contrairement au problème de la sous-section suivante.

5.4.3 Points de repère dans le temps

Le second problème ayant trait aux points de repère est que, dans la réalité, on ne peut savoir combien de points de repère différents les plates-formes rencontreront.

Par exemple, une plate-forme peut naviguer dans un environnement inconnu en ligne droite pendant une heure. Vers la fin du trajet, les points de repère détectés lors des premières itérations ne seront naturellement plus visibles depuis longtemps. Il ne sert alors à rien de garder leurs positions à l'intérieur du vecteur d'état.

Parallèlement, à l'initialisation, aucun point de repère n'a pu être aperçu encore. Il faut donc également avoir la possibilité d'ajouter une position à l'intérieur du vecteur d'état.

Ces modifications au vecteur d'état impliquent des difficultés non seulement infor-

matiques, mais également théoriques. Le fait que le nombre de points de repère M change implique dans notre cas que le nombre de variables de bruit de processus change également. Les dimensions n et q deviennent variables ainsi que les dimensions de $\mathbf{x}^k$, $\mathbf{w}^k$, $\mathbf{A}^k$, $\mathbf{W}^k$, $\mathbf{K}^k$, $\mathbf{P}^k$ et $\mathbf{f}$.

Le problème théorique survient à l'ajout au vecteur d'état de la position d'un point de repère nouvellement détecté. À ce moment précis, l'ancienne matrice de covariance de l'erreur d'estimation $\mathbf{P}^{k-1}$, qui est indispensable dans les équations de l'itération k, devient invalide, car elle a les mauvaises dimensions.

Il faut donc, pour chaque point de repère nouvellement détecté, ajouter deux variables dans le vecteur d'état ainsi que deux colonnes et deux lignes dans la matrice de covariance de l'erreur d'estimation. Le problème est le même que lors de l'initialisation du filtre de Kalman : il faut donner une valeur initiale à ces nouvelles entrées.

Toutefois, dans ce cas-ci, il est impossible de supposer connue la position d'un nouveau point de repère, car il s'agit de sa position relative à la plate-forme à une itération quelconque, et non à l'initialisation. Par conséquent, il faut estimer les nouvelles valeurs du vecteur d'état et de la matrice de covariance de l'erreur d'estimation.

En inversant les équations du modèle de mesure et en supposant que l'on puisse utiliser la donnée de distance (grâce à l'hypothèse de sol plat et de points de repère au sol), il est possible d'obtenir la position approximative d'un point de repère en fonction de sa position dans l'image omnidirectionnelle. Cette position sert d'initialisation au vecteur d'état lors de la première détection de ce point de repère.

Pour ce qui est de la matrice de covariance de l'erreur d'estimation, on utilise

les mêmes valeurs que lors de l'initialisation du filtre, comme expliqué à la sous-section 5.4.1.

5.4.4 Problème d'intervalle d'angles

Plusieurs vecteurs du système contiennent des angles pour variables. Lorsque le filtre de Kalman fait des opérations avec ces variables, il les additionne ou les soustrait comme n'importe quel nombre. Cependant, un angle α a la propriété particulière que $\alpha = \alpha + a \cdot 2\pi, \quad a \in \mathbb{N}$.

Cette propriété peut causer des problèmes. Prenons par exemple l'équation de correction de l'état (5.29). On y soustrait angles réels et angles mesurés. Cette différence est ensuite multipliée par le gain de Kalman et sert directement de correction à l'état.

S'il y a un décalage de 2π pour un de ces angles, ce décalage sera amplifié par le gain de Kalman et il y aura une correction beaucoup plus grande que prévu. La solution pour éviter ces problèmes est d'utiliser l'opérateur $\ominus$ tel que décrit à la sous-section 5.2.1 plutôt que la soustraction habituelle lorsque vient le temps de comparer des angles.

5.4.5 Implantation informatique du filtre

Dans le cadre de ce projet de maîtrise, le rôle joué par le filtre de Kalman est à ce point primordial qu'une librairie de classes `C++` implantant le filtre de Kalman a été conçue. Une librairie de filtrage de Kalman, étendu ou linéaire, à dimensions variables est maintenant disponible au département de génie électrique de l'École Polytechnique de Montréal. Elle est le fruit du travail de l'auteur, mais il faut

souligner la supervision du professeur Richard Gourdeau (qui a fait une première implantation du filtre étendu à dimensions fixes en `C`) et la collaboration de son étudiant Sylvain Marleau pour le développement et la phase de test.

Cette librairie est assez simple d'utilisation pour tester rapidement un filtre de Kalman sans se préoccuper des équations de gain, de prédiction et de correction. De bonnes fonctions de déverminage ont été incluses pour faciliter le développement. La souplesse et les possibilités d'optimisation n'ont pas été sacrifiées pour autant. Il est possible de prédire des états passés, présents ou futurs, de changer n'importe quelles dimensions et même de modifier certains calculs intermédiaires du filtre, ce qui est pratique pour utiliser l'opérateur $\ominus$, par exemple.

Certains calculs peuvent être factorisés pour gagner du temps d'exécution. Si certaines matrices ont la propriété d'être diagonales, il est possible de le spécifier afin que la librairie optimise les calculs en tirant profit des propriétés de ces matrices.

Finalement, la décomposition UDU (parente à la décomposition de Cholesky) de la matrice de covariance de l'erreur d'estimation a été utilisée pour les calculs internes (Bierman, 1977). La matrice $\mathbf{P}^k$ est par définition symétrique et définie positive. Or, il est possible, à cause des erreurs d'arrondis de l'ordinateur, que cette symétrie soit altérée au fur et à mesure des prédictions et des corrections. Une telle anomalie peut causer des erreurs croissantes d'estimation de l'état.

La matrice est décomposée ainsi : $\mathbf{P} = \mathbf{U}\mathbf{D}\mathbf{U}^{\mathrm{T}}$, où $\mathbf{U}$ est triangulaire supérieure avec diagonale unitaire et où $\mathbf{D}$ est diagonale. En opérant sur ces matrices, la symétrie et la propriété définie positive sont assurées. Il est démontré qu'en utilisant cette méthode avec des valeurs réelles sur 32 bits, on obtient une précision équivalente à celle d'un filtre de Kalman normal utilisant une représentation sur 64 bits.

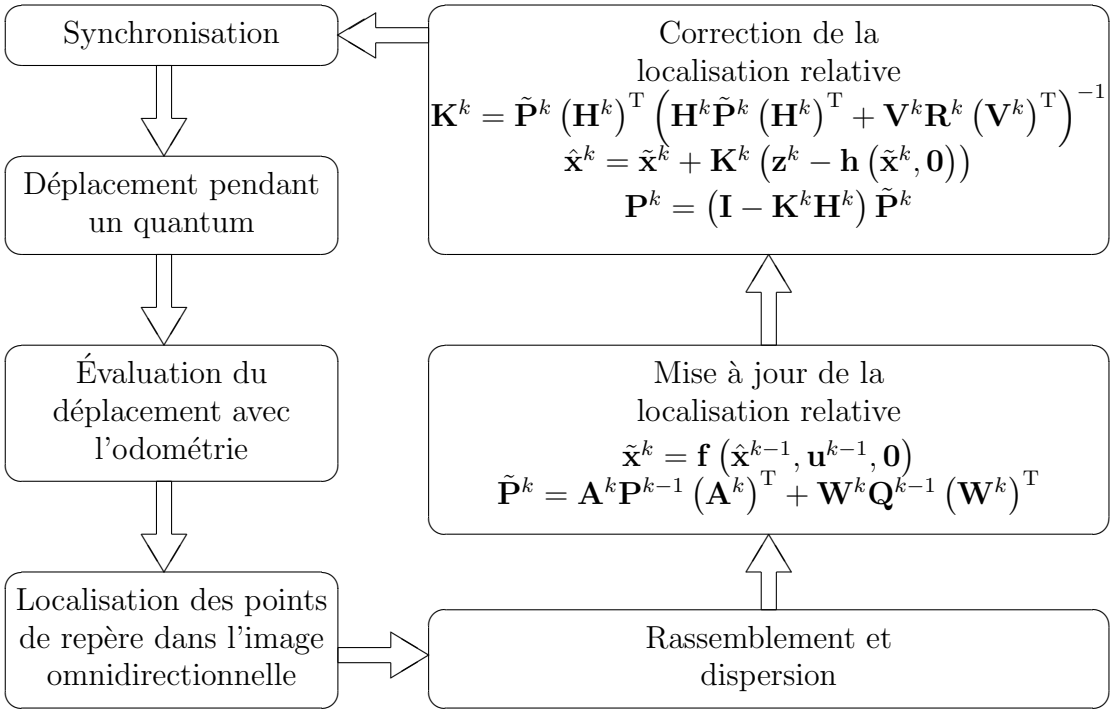


FIG. 5.3 Schéma bloc du fonctionnement du filtre de Kalman

5.5 Conclusion

Ce chapitre a présenté les équations nécessaires pour mettre à jour et corriger une estimation de la localisation relative grâce au filtrage de Kalman. Le schéma bloc de la figure 5.3 montre les opérations à effectuer dans l'ordre à l'intérieur d'un quantum.

La théorie du filtrage de Kalman étendu propre à ce travail n'est pas innovatrice en soi. Cependant, la modélisation mathématique du processus ainsi que de la prise de mesure est le travail de l'auteur. De plus, la généralisation au cas d'un filtre à dimension variable est une contribution de ce projet au domaine du filtrage de Kalman.

CHAPITRE 6

ANALYSE DES RÉSULTATS

Ce chapitre décrit l'expérimentation de la méthode de localisation relative mutuelle proposée et présente les résultats obtenus. Il donne tout d'abord une courte analyse du nombre minimal théorique de points de repère afin d'assurer la convergence de la méthode d'estimation de la localisation relative développée dans le cadre du projet. Ensuite, des résultats de simulation sont présentés, suivis d'une discussion de l'influence de différents facteurs sur la qualité de l'estimation. Des résultats obtenus avec des plates-formes réelles sont ensuite analysés. Le chapitre se termine sur une discussion à propos de la robustesse du logiciel développé.

6.1 Nombre minimal de points de repère

Il est intéressant de connaître le nombre minimal théorique de points de repère pour que le système à résoudre par filtrage de Kalman ne soit pas sous-déterminé. Dans ce cas, les mesures à elles seules suffiraient, en théorie, pour connaître la configuration relative des plates-formes, sans même avoir à se fier à l'estimation précédente de l'état et aux estimations de déplacements. Dans le cas contraire, il y aurait une infinité de localisations relatives ne violant aucune des mesures prises, ce qui laisse présager une mauvaise convergence de l'algorithme de Kalman. Nous présentons ici une analyse simple du nombre d'équations et d'inconnues en fonction du nombre de plates-formes mobiles N, du nombre de points de repère M et des informations disponibles pour chaque détection : orientations seulement ou avec distances approximatives.

6.1.1 Avec mesures de distance

Le nombre d'inconnues du système est en fait le nombre de variables dans le vecteur d'état. Comme indiqué à la sous-section 5.3.2.1, il y a donc $3(N-1)+2M$ inconnues : trois pour chaque pose de plate-forme (excepté la première plate-forme, qui se trouve à l'origine du repère mobile par définition) et deux pour chaque position de point de repère.

Nous supposerons, pour simplifier l'analyse, que toutes les plates-formes détectent tous les points de repère. Comme chaque détection retourne une orientation et une distance, on obtient $2MN$ équations, tel qu'indiqué à la sous-section 5.3.2.5. Les équations du modèle de processus (sous-section 5.3.2.4) ne peuvent être comptées ici, car elles ne permettent pas d'ajouter de contraintes à un état courant, seulement de mettre à jour un ancien état.

Pour ne pas avoir de système sous-déterminé, il faut que le nombre d'équations soit au moins égal, sinon supérieur au nombre d'inconnues :

$$2MN \geq 3(N-1) + 2M \tag{6.1}$$

ce qui, après quelques manipulations, devient :

$$M > \frac{3}{2} \tag{6.2}$$

Par conséquent, on peut conclure que 2 points de repère détectés en permanence par toutes les plates-formes suffisent à assurer qu'un système ne soit pas sous-déterminé. Ce qui est intéressant, c'est que ce nombre de points de repère ne dépend pas du nombre de plates-formes mobiles. Cela s'explique simplement par le fait que chaque nouvelle plate-forme ajoute 3 inconnues, mais $2M$ équations.

Cette conclusion ne signifie pas que le filtre de Kalman ne peut pas prédire correctement la localisation relative mutuelle des plates-formes si un seul point de repère est disponible. Au contraire, si l'estimation actuelle de la localisation est bonne et que les données odométriques ne sont pas trop erronées, il est possible de conserver une excellente estimation de la localisation, même si un seul point de repère est visible par les plates-formes. Toutefois, la correction n'assure peut-être plus la convergence vers la réalité : si l'état actuel estimé est trop différent de la réalité, alors le filtre pourrait converger vers une autre solution satisfaisant toutes les contraintes du système d'équations. Une étude d'observabilité devient alors nécessaire pour évaluer la convergence du filtre.

6.1.2 Sans mesure de distance

S'il est impossible de supposer le sol plat ou le contact des points de repère avec le sol, alors seules les orientations des points de repère peuvent être mesurées. Le nombre d'inconnues ne change pas, mais il y a deux fois moins d'équations :

$$MN \geq 3(N-1) + 2M \tag{6.3}$$

ce qui, après quelques manipulations, devient :

$$M > 3 + \frac{3}{N-2}, \qquad N > 2 \tag{6.4}$$

Selon cette inéquation, le nombre de points de repère nécessaires diminue lorsque le nombre de plates-formes augmente, jusqu'à une limite de 4 points de repère pour 5 robots et plus. La mauvaise nouvelle est pour un groupe de 2 robots : dans ce cas, il ne peut jamais y avoir assez de points de repère pour que le système ne soit pas sous-déterminé à un instant donné, car chaque point de repère ajoute 2 équations

TAB. 6.1 Nombre minimal théorique de points de repère

N	M_1	M_2
2	2	—
3	2	6
4	2	5
5+	2	4

et 2 inconnues. Toutefois, cela ne signifie pas que la convergence de l'algorithme est impossible, comme nous allons maintenant en discuter.

6.1.3 Conclusion

Le tableau 6.1 résume le nombre minimal théorique de points de repère pour que le système d'équations ne soit pas sous-déterminé à un instant donné, M_1 étant le nombre minimal si les distances sont mesurées et M_2, si elles ne le sont pas.

La méthode de localisation implantée dans le cadre de ce projet est donc très performante si l'on peut supposer le sol plat et les points de repère en contact avec le sol. Deux détections en commun suffisent pour assurer une convergence de l'estimation vers la réalité.

Par contre, la méthode semble peu adaptée si ces hypothèses ne tiennent pas, et peu fonctionnelle pour deux plates-formes mobiles. Dans ce cas, il serait préférable d'utiliser la stéréographie omnidirectionnelle afin d'estimer la distance des points de repère détectés dans le but de se libérer des hypothèses contraignantes émises précédemment.

Comme seules deux plates-formes munies de caméras omnidirectionnelles étaient disponibles au moment des essais, nous n'avons pas testé la méthode en n'utilisant pas les mesures de distance. Par conséquent, le reste de ce chapitre se concentre à

présenter les résultats de la méthode utilisant les mesures de distance des points de repère obtenues grâce à l'hypothèse de sol plat avec contact des points de repère.

Il est important de signaler que les conditions de convergence exprimées ici sont suffisantes, mais non nécessaires. Il est possible que l'algorithme converge avec un nombre de points de repère inférieur à celui indiqué. Une étude d'observabilité devrait être effectuée afin de connaître plus en détail la capacité de convergence du filtre de Kalman selon les conditions d'un problème donné. Néanmoins, les nombres donnés ici sont un bon indicateur de la capacité de convergence de l'algorithme développé.

6.2 Conditions de simulation

La méthode développée dans le cadre de ce projet permet de calculer la pose d'une plate-forme dans le repère mobile centré sur une autre plate-forme. Afin de valider les résultats, il est nécessaire de connaître réellement cette pose afin de la comparer à l'estimation. Elle peut être calculée à partir de la pose des deux plates-formes dans le repère absolu de l'environnement.

Cependant, sans système de positionnement absolu, il est difficile d'obtenir avec précision la pose absolue des plates-formes dans la réalité. Par conséquent, le simulateur *Stage* (le pilote de simulation de l'architecture *Player* — voir sous-section 3.2.2) a été utilisé afin d'obtenir les poses absolues et calculer les poses relatives réelles à comparer aux estimations.

La figure 6.1 montre la scène simulée utilisée lors des tests, à l'échelle d'un mètre par case. Elle contient trois points de repère, orange, vert et bleu, situés respectivement en $(-2.0, 0.2)$, $(0.0, 2.4)$ et $(3.0, -2.0)$. Les plates-formes se déplacent selon une trajectoire circulaire de rayon 1.5 m centrée en $(0.5, 0.0)$, à une vitesse de 0.15 m/s.

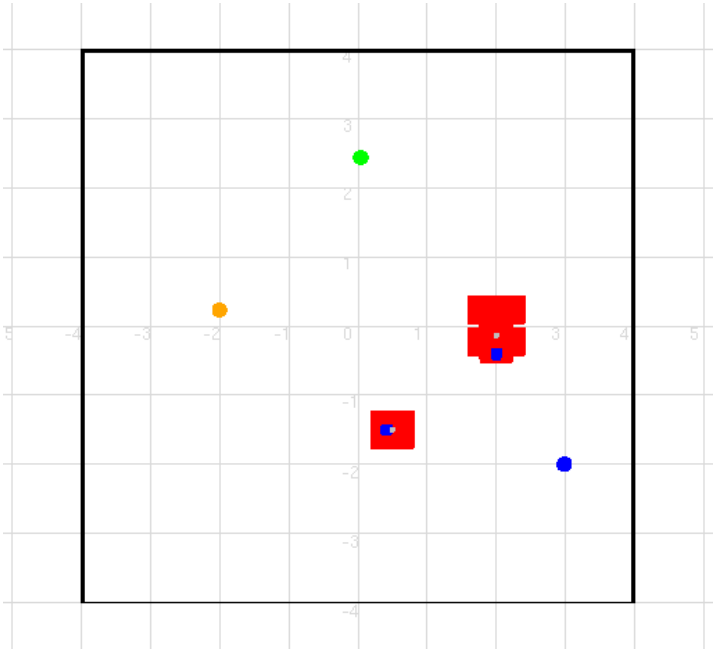


FIG. 6.1 Scène simulée avec *Stage*

Le simulateur *Stage* permet de bruiter l'odométrie des plates-formes. Lors des simulations, un bruit maximal de 5% a été appliqué en translation et en rotation pour les plates-formes mobiles. Le bruit de mesure a été simulé avec un modèle de bruit gaussien. Un bruit avec écart-type de 2° a été appliqué à l'orientation des points de repère, alors qu'un bruit avec écart-type de 3 pixels a été appliqué à la distance radiale des points de repère dans l'image omnidirectionnelle.

Le filtre de Kalman a été initilialisé lors des essais avec une estimation grossière de la localisation relative des plates-formes, sans aucune information *a priori* au sujet des points de repère. Cette estimation comportait généralement une erreur d'environ 1 m en distance et environ 45° en orientation.

Cette estimation grossière semble réaliste, car la configuration relative de plates-formes au début d'une tâche est souvent connue approximativement. Dans le cas contraire, il faut recourir à une méthode de localisation relative non récursive des

plates-formes afin de procurer au filtre de Kalman une estimation initiale.

Toutefois, les résultats d'un test présenté plus loin dans ce chapitre démontrent que la méthode développée peut fonctionner sans même avoir d'estimation initiale valide. Seules les quelques premières itérations suffisent à l'algorithme pour converger vers la configuration réelle des plates-formes et points de repère.

Enfin, il est important de spécifier que le quantum utilisé pour le filtre de Kalman est de 0.4 s ; chaque seconde comporte donc deux itérations et demie.

6.3 Résultats de simulation

La qualité de l'estimation de la localisation relative produite par la solution développée peut être évaluée par la distance entre la position réelle d'une plate-forme dans le repère mobile et l'estimation de cette position dans le même repère. La figure 6.2 donne les résultats d'une simulation ayant les paramètres décrits à la section 6.2.

L'orientation peut être évaluée de la même façon, en calculant la différence entre l'orientation réelle et l'orientation estimée de la plate-forme. La figure 6.3 présente les résultats au cours de la même simulation.

Accessoirement, la méthode permet également de déterminer la position des points de repère utilisés afin de corriger l'estimation de la localisation relative mutuelle. Il est intéressant de comparer la position réelle d'un point de repère dans le repère mobile d'une plate-forme avec la position estimée, en calculant la distance entre les deux positions, comme présenté à la figure 6.4.

Ces résultats démontrent clairement la rapidité de convergence de l'algorithme. En

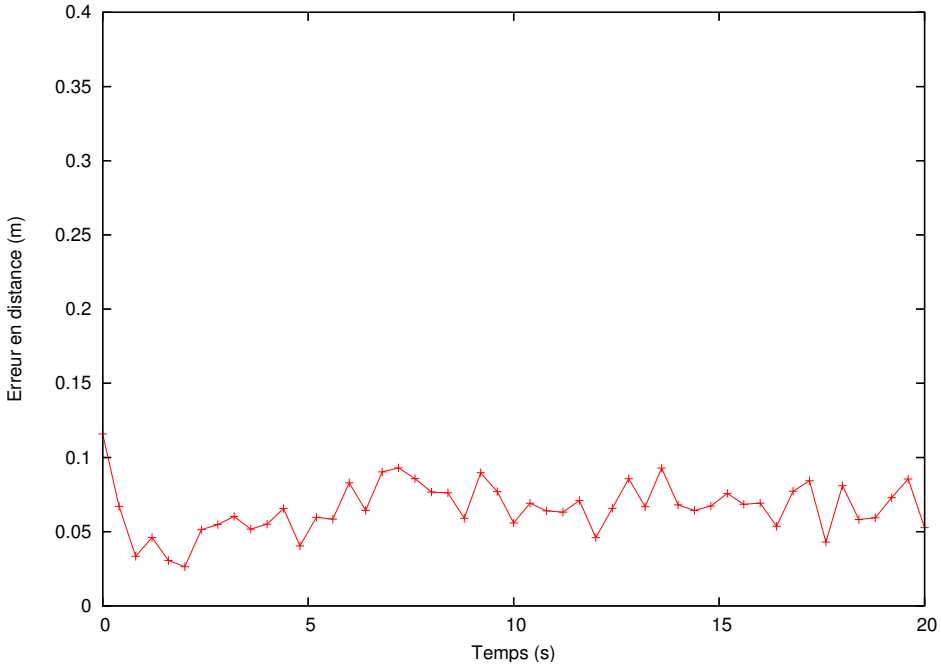


FIG. 6.2 Erreur de position d'une plate-forme

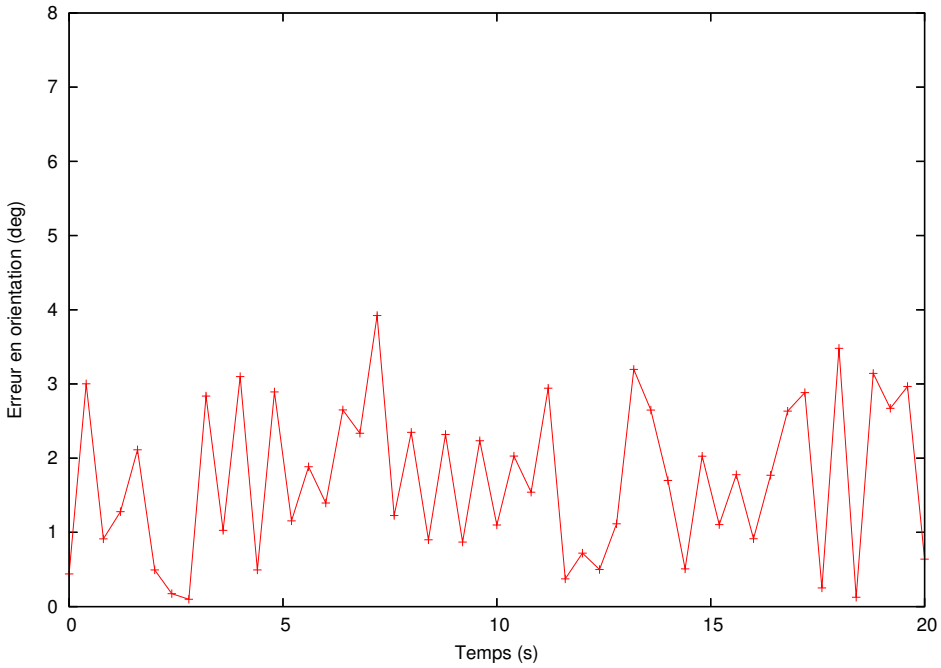


FIG. 6.3 Erreur d'orientation d'une plate-forme

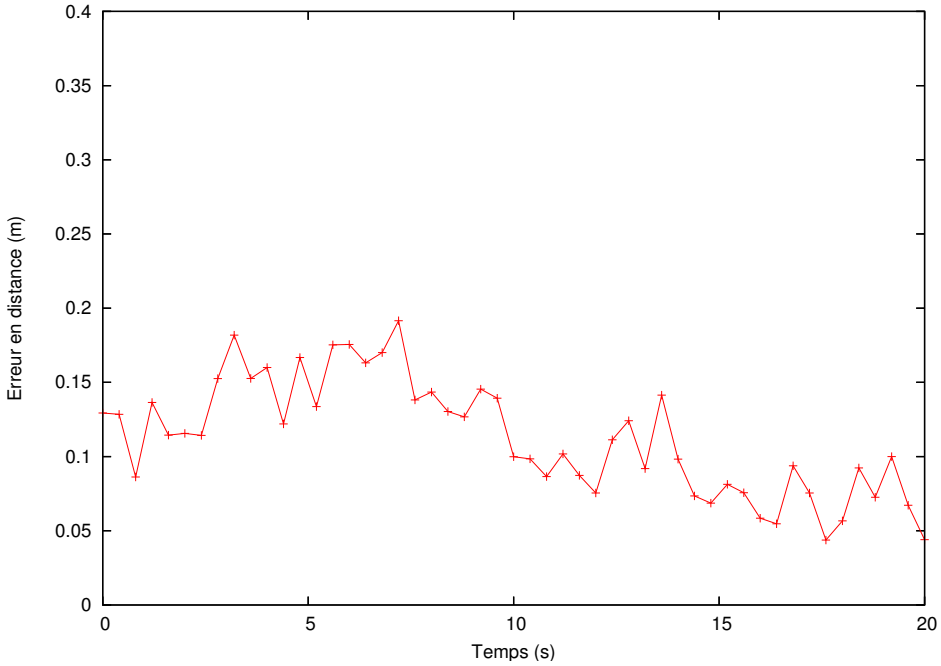


FIG. 6.4 Erreur de position d'un point de repère

deux itérations seulement, l'erreur d'estimation atteint déjà sa valeur moyenne. Ainsi, à partir d'une estimation grossière (avec environ 1 m et 45° d'erreur), moins d'une seconde suffit à l'algorithme pour se stabiliser.

La méthode permet d'obtenir, en simulation, une estimation de la position d'un point de repère entachée d'une erreur inférieure à 10 cm. L'erreur d'orientation, quant à elle, est beaucoup plus variable, mais dépasse rarement 4°.

Enfin, même si la méthode développée dans le cadre du projet n'avait pas pour but premier de localiser des points de repère, on note que l'estimation en position est tout de même précise à 20 cm près. Il est intéressant de remarquer que, contrairement à la position d'une plate-forme, la position d'un point de repère tend à se préciser davantage avec le temps : après 15 s, l'erreur ne dépassait plus 10 cm. Cela est dû en partie au fait que le point de repère ne se déplace pas, et ne subit donc pas l'erreur d'odométrie des plates-formes.

6.4 Analyse des facteurs influençant la méthode

Cette section discute de l'importance de différents facteurs sur la qualité de l'estimation obtenue par le filtre de Kalman. Les facteurs étudiés sont : le nombre de plates-formes, le nombre de points de repère, la qualité de l'initialisation, la vitesse de déplacement et la quantité de bruit.

Afin de faciliter la comparaison des différents facteurs entre eux, l'échelle des graphiques d'erreur présentés est la même que celle des graphiques présentés à la section 6.3, sauf avis contraire.

6.4.1 Influence du nombre de plates-formes

Lorsque le nombre de plates-formes augmente, est-ce que la qualité de l'estimation s'améliore ou se détériore ? Des plates-formes supplémentaires impliquent de nouvelles sources de bruit de processus et de bruit de mesure pouvant ternir la précision des résultats. Toutefois, chaque nouvelle plate-forme permet de potentiellement générer M nouvelles détections. Ainsi, chaque nouvelle plate-forme ajoute 3 inconnues, mais $2M$ équations (6, dans cette simulation).

Les figures 6.5, 6.6 et 6.7 montrent respectivement l'erreur de position, d'orientation d'une plate-forme ainsi que l'erreur de position d'un point de repère dans le cas de deux et de trois plates-formes.

Les commentaires précédents laissent présager une amélioration de l'estimation avec l'augmentation du nombre de plates-formes. Cependant, les courbes ne permettent pas d'arriver à une telle conclusion. En fait, il semble que le nombre de plates-formes n'ait que peu d'influence sur les résultats obtenus.

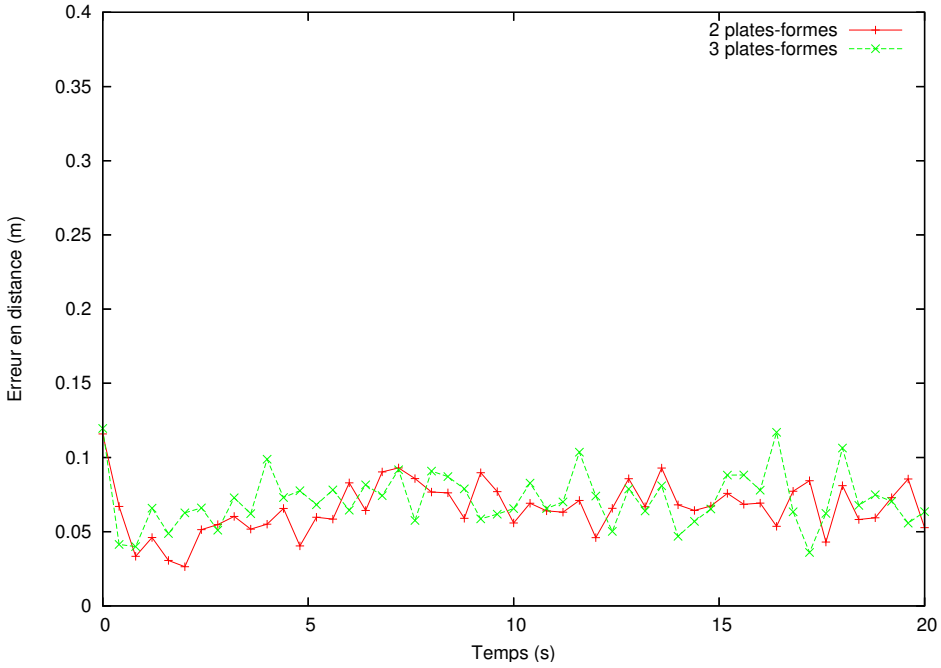


FIG. 6.5 Erreur de position en fonction du nombre de plates-formes

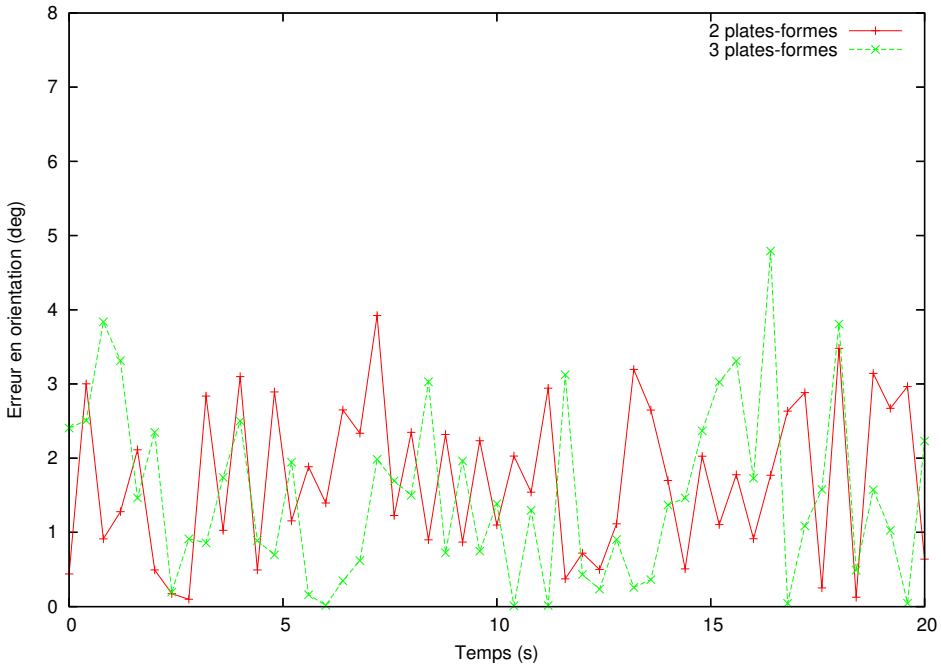


FIG. 6.6 Erreur d'orientation en fonction du nombre de plates-formes

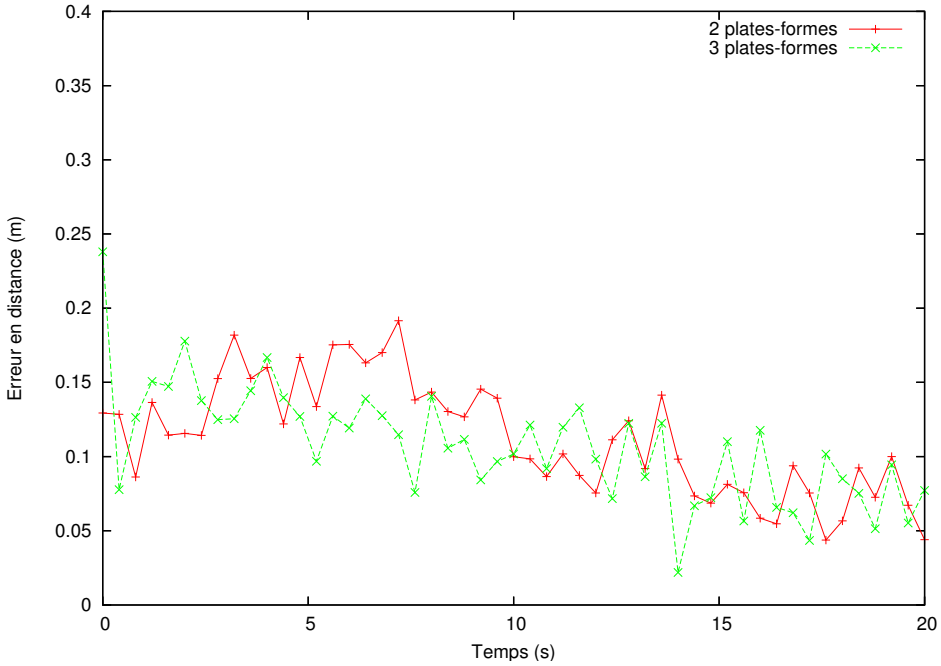


FIG. 6.7 Erreur du point de repère en fonction du nombre de plates-formes

Il est possible qu'en simulation, deux plates-formes et trois points de repère suffisent à obtenir la meilleure estimation possible, compte tenu du bruit. Alors, le fait d'ajouter une plate-forme ne peut améliorer davantage les résultats.

Des tests réels avec plusieurs plates-formes sont nécessaires afin d'éclaircir davantage cette question.

6.4.2 Influence du nombre de points de repère

Comme les points de repère sont les outils servant à corriger l'estimation de la localisation relative, il est naturel de penser que la précision de l'estimation augmentera avec le nombre de points de repère. Comme discuté à la sous-section 6.1.1, chaque point de repère permet d'ajouter jusqu'à $2N$ équations (4, dans ce cas-ci) pour seulement 2 inconnues.

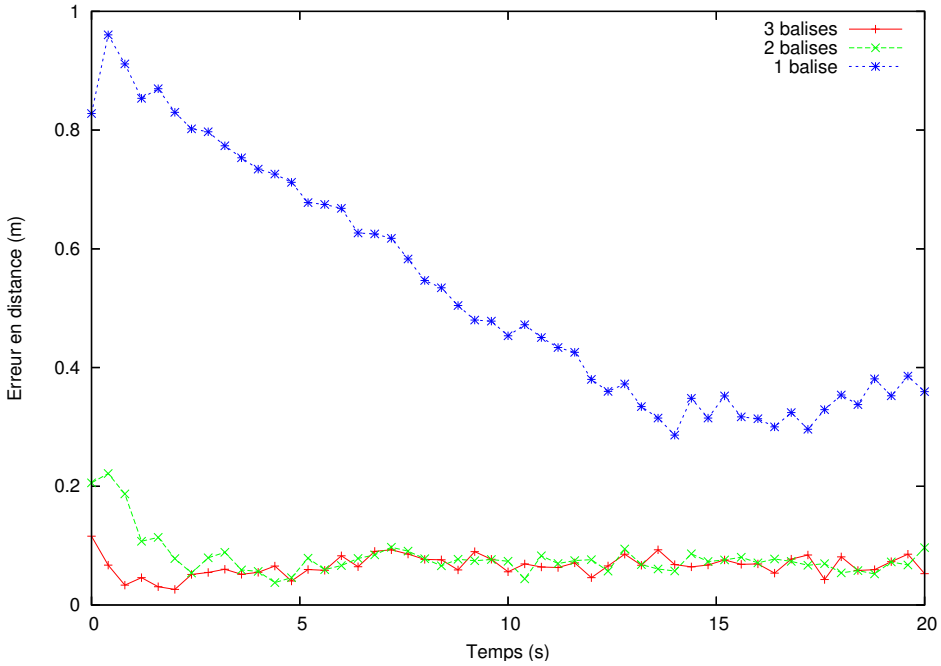


FIG. 6.8 Erreur de position en fonction du nombre de points de repère

Les figures 6.8, 6.9 et 6.10 montrent les résultats du test en fonction du nombre de balises utilisées comme points de repère : une, deux ou trois balises. Les échelles de ces trois graphiques ne sont pas les mêmes que les échelles des autres graphiques, afin de bien montrer les erreurs encourues si un seul point de repère est détecté.

Pour l'erreur de position d'une plate-forme, on note que le fait d'avoir deux points de repère ne change que très peu la valeur atteinte en régime permanent par rapport à trois points de repère. Toutefois, la quantité de points de repère permet de converger plus rapidement vers cette solution : avec 2 balises, environ 2.4 s sont nécessaires, alors qu'avec 3 balises, la convergence est pratiquement immédiate.

Il est intéressant de remarquer que, comme le prédisait l'analyse du nombre d'équations et d'inconnues, le nombre minimal de points de repère permettant d'assurer une bonne convergence est de deux. Avec une seule balise, plus de 12 s sont nécessaires afin de descendre sous la barre des 40 cm d'erreur, contrairement à 1 s pour

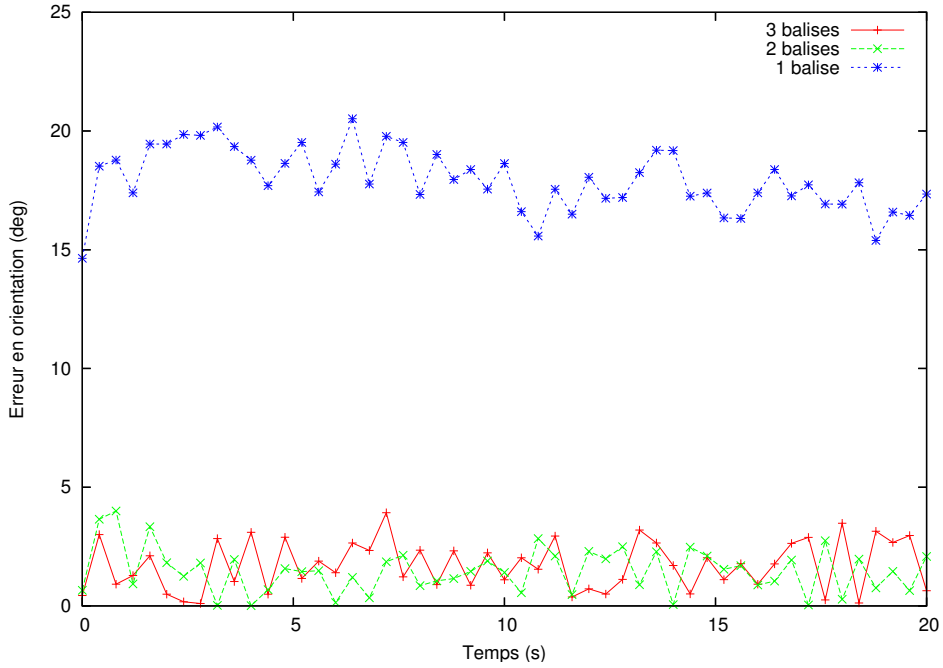


FIG. 6.9 Erreur d'orientation en fonction du nombre de points de repère

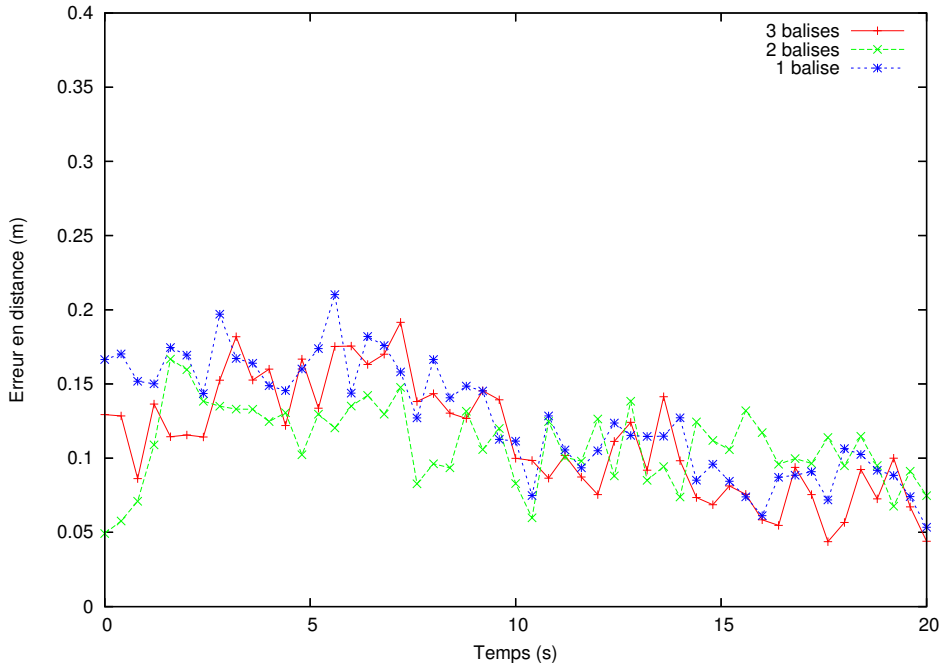


FIG. 6.10 Erreur du point de repère en fonction du nombre de points de repère

la barre des 10 cm d'erreur si plus d'une balise est disponible. L'erreur de position semble converger malgré tout, quoique très lentement. Cette convergence dépend fortement de la qualité de l'estimation initiale fournie au filtre de Kalman.

Le nombre de points de repère n'a pas influencé autant l'erreur d'orientation, qui se maintient en-deça de 4°, avec 2 ou 3 balises. Par contre, avec une seule balise, l'erreur frôle les 20° sans vouloir converger vers un meilleur état.

Enfin, contrairement aux erreurs de localisation de plates-formes, l'erreur de position d'un point de repère en particulier n'est pas affectée autant par le nombre total de points de repère, ce qui est logique. Les points de repère supplémentaires ajoutent des équations impliquant les plates-formes et non pas les autres points de repère.

6.4.3 Influence de la qualité de l'initialisation

L'estimation initiale de la localisation relative fournie au filtre de Kalman influence grandement les premières itérations du filtre. Ainsi, si l'initialisation du filtre est bonne, l'estimation devrait converger plus rapidement vers la réalité.

Les figures 6.11, 6.12 et 6.13 présentent l'influence de la qualité de l'initialisation du filtre de Kalman sur la qualité de l'estimation obtenue. Pour générer ces graphiques, trois initialisations différentes ont été utilisées. L'initialisation « parfaite » consiste en fait à fournir au filtre de Kalman, dès la première itération, la localisation relative mutuelle exacte des plates-formes. L'initialisation dite « approximative » est celle utilisée tout au long de ces tests : environ 1 m d'erreur en distance et environ 45° d'erreur en orientation. Enfin, « aucune » initialisation signifie que toutes les plates-formes ont été initialement positionnées en $(0, 0, 0)$.

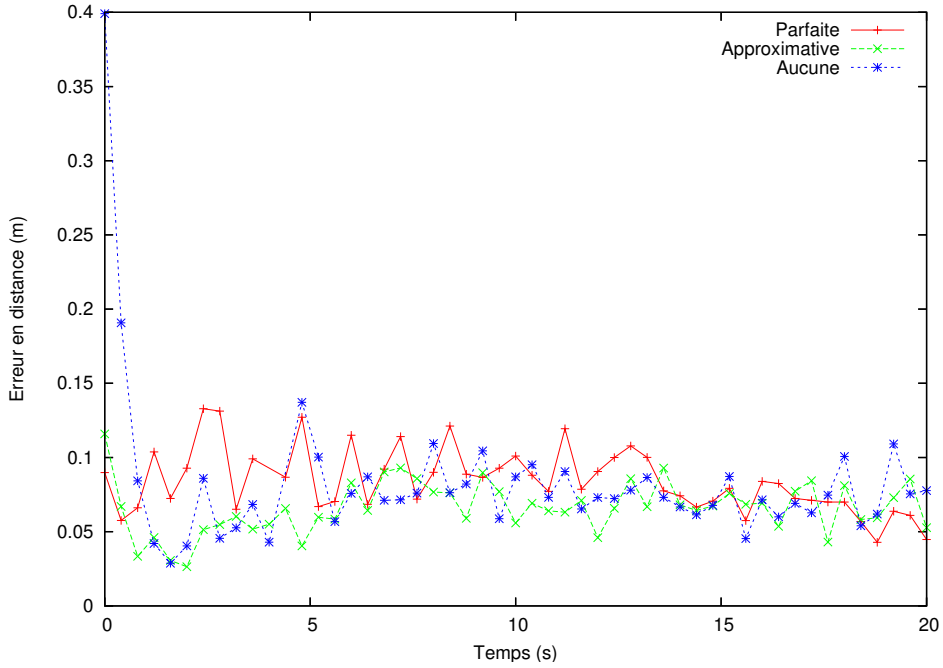


FIG. 6.11 Erreur de position en fonction de la qualité de l'initialisation

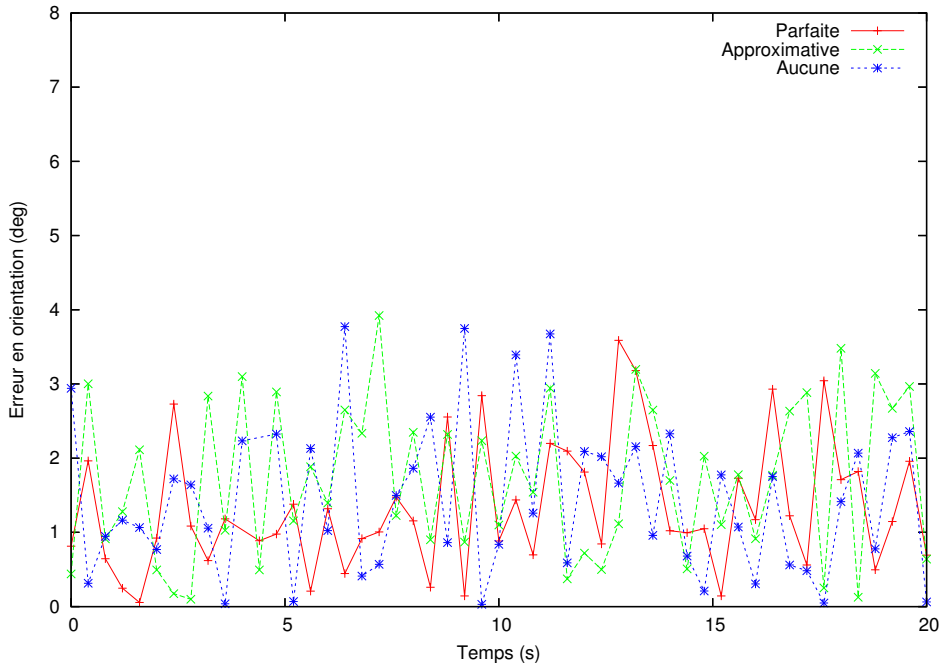


FIG. 6.12 Erreur d'orientation en fonction de la qualité de l'initialisation

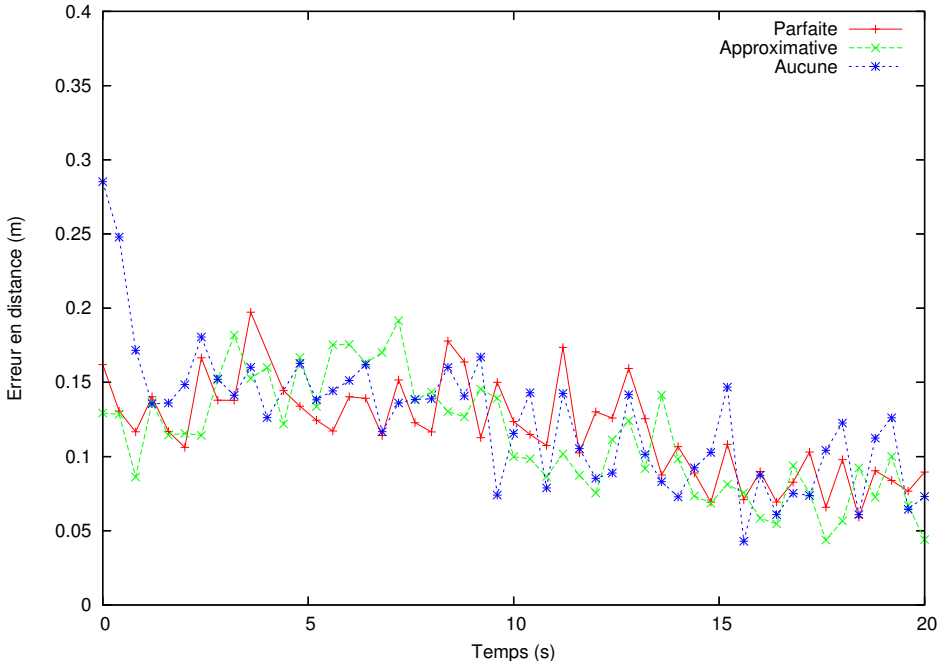


FIG. 6.13 Erreur du point de repère en fonction de la qualité de l'initialisation

On remarque, dans les deux graphiques d'erreur en position, que l'estimation semble effectivement converger plus rapidement si l'initialisation du filtre se rapproche de la réalité. Si aucune initialisation n'est faite, les deux ou trois premières itérations du filtre donnent de moins bons résultats.

Cela démontre tout de même qu'en simulation, il n'est pas nécessaire d'avoir de méthode supplémentaire de localisation relative non récursive pour initialiser le filtre de Kalman : 1 s suffit pour obtenir une bonne estimation de la localisation.

6.4.4 Influence de la vitesse de déplacement

Dans la réalité, plus une plate-forme se déplace rapidement, plus les bruits de processus et de mesure augmentent. Dans la simulation, la qualité de l'odométrie se détériore à mesure que la vitesse augmente. Toutefois, dans la réalité, les vibra-

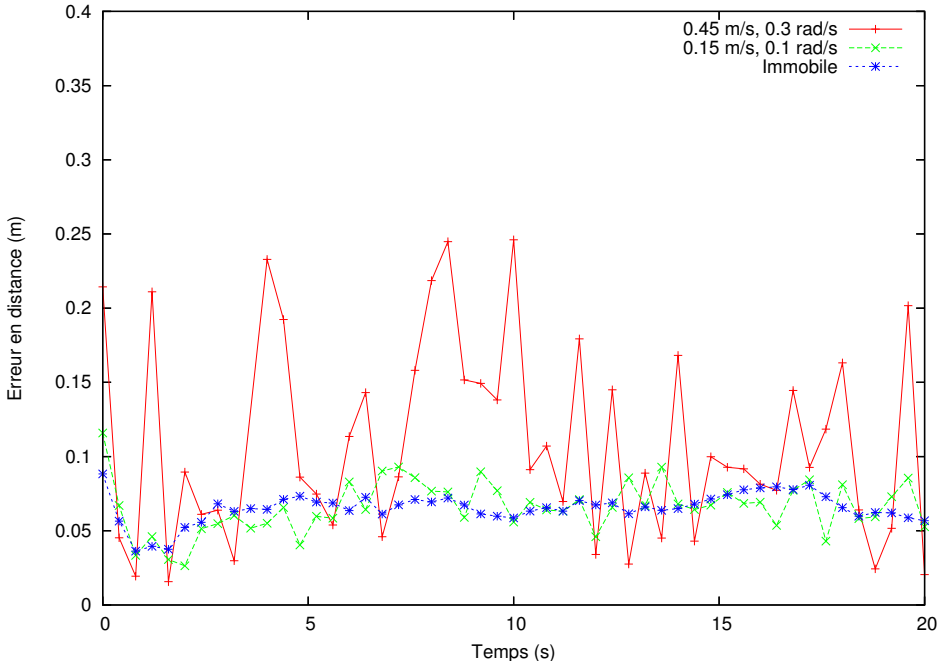


FIG. 6.14 Erreur de position en fonction de la vitesse

tions supplémentaires induiront également de plus grandes erreurs de localisation dans l'image omnidirectionnelle. Ces erreurs de mesure supplémentaires ne sont pas modélisées dans la simulation.

Les résultats de la localisation relative pour des plates-formes se déplaçant rapidement (0.45 m/s), lentement (0.15 m/s) ou étant immobiles sont présentés aux figures 6.14, 6.15 et 6.16.

Un plus grand bruit signifie normalement une plus grande variabilité des estimations obtenues ; c'est en effet la conclusion à tirer à partir des graphiques de résultats. La vitesse est de loin le facteur qui influence le plus la qualité des résultats obtenus.

Lorsque les plates-formes sont immobiles, seules les erreurs de vision subsistent. Les résultats deviennent aussitôt très stables : une erreur de position de plate-forme

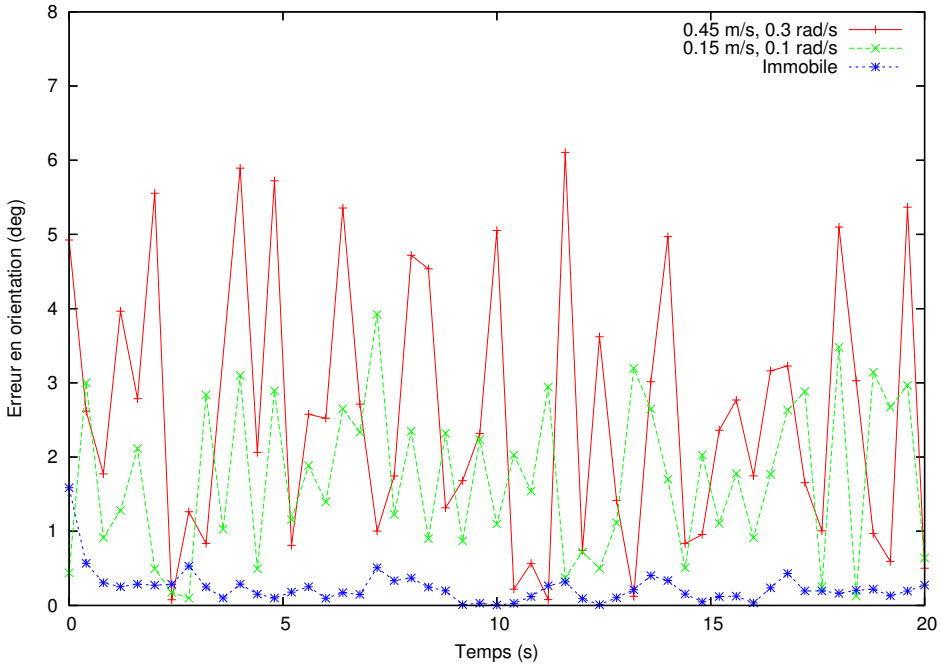


FIG. 6.15 Erreur d'orientation en fonction de la vitesse

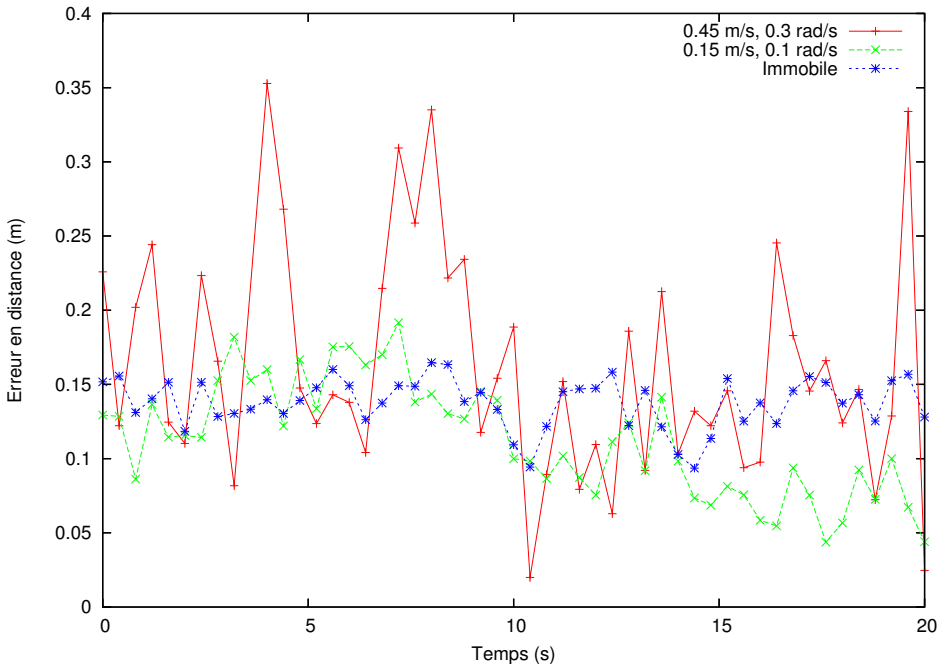


FIG. 6.16 Erreur du point de repère en fonction de la vitesse

entre 4 et 8 cm, une erreur d'orientation inférieure à 1° et une erreur de position d'un point de repère entre 9 et 16 cm. Toutefois, lorsque l'on triple la vitesse de la plate-forme par rapport à la vitesse utilisée pour les autres tests, les résultats deviennent beaucoup plus variables : jusqu'à 25 cm d'erreur pour la position de plate-forme, 6° en orientation et 35 cm pour la position de point de repère.

Par conséquent, il peut être bénéfique pour une plate-forme se fiant sur la méthode de localisation relative mutuelle développée de ralentir par moments ou même de s'immobiliser afin d'augmenter la précision des résultats obtenus.

6.4.5 Influence de la quantité de bruit

Plus le bruit de mesure de la caméra omnidirectionnelle et l'imprécision de l'odométrie sont importants, moins l'estimation de la localisation relative sera précise. Les figures 6.17, 6.18 et 6.19 corroborent cette affirmation.

Il s'agit là encore d'un facteur d'importance. En doublant le bruit, l'erreur moyenne en position passe de 8 cm à 15 cm. Il est intéressant de remarquer que, plus le temps avance, plus le filtre de Kalman tend à contre-balancer l'effet nocif du bruit. Par exemple, l'erreur maximale de position d'un point de repère est d'environ 35 cm avec double bruit, alors qu'elle est de 20 cm avec le bruit normal. Or, après 20 s, l'erreur est descendue à environ 10 cm dans les deux cas.

Cependant, il est plutôt inattendu de voir que le bruit n'a que peu d'influence sur l'erreur d'orientation d'une plate-forme. Plus d'expérimentations à ce sujet permettraient de comprendre davantage ce comportement.

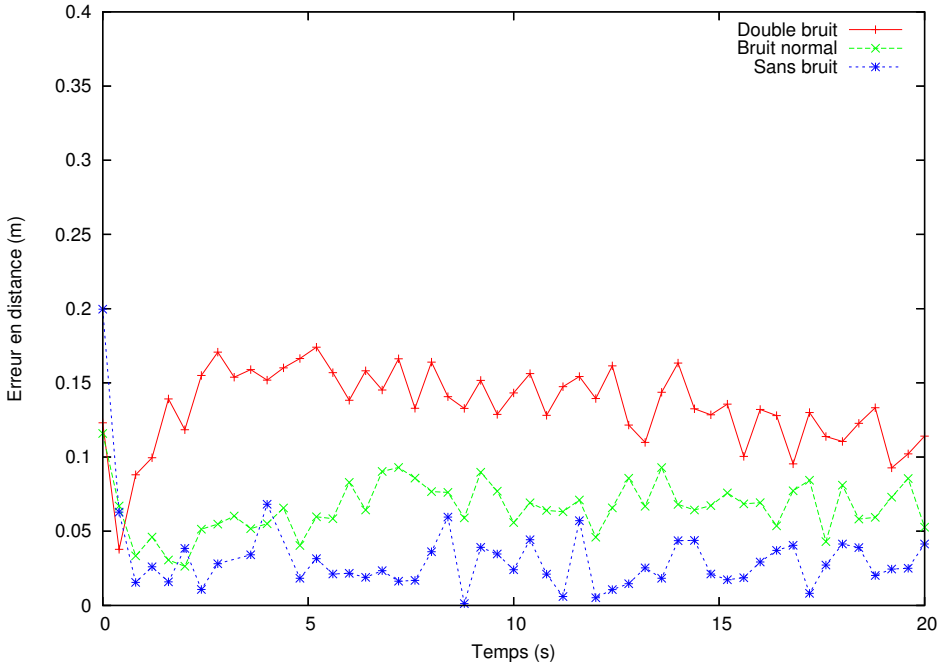


FIG. 6.17 Erreur de position en fonction du bruit

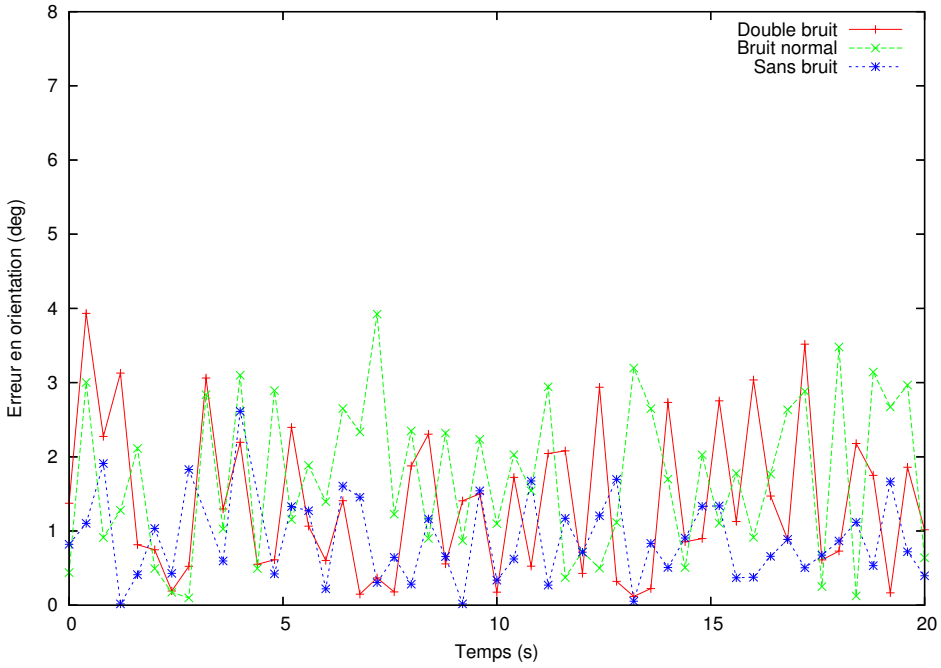


FIG. 6.18 Erreur d'orientation en fonction du bruit

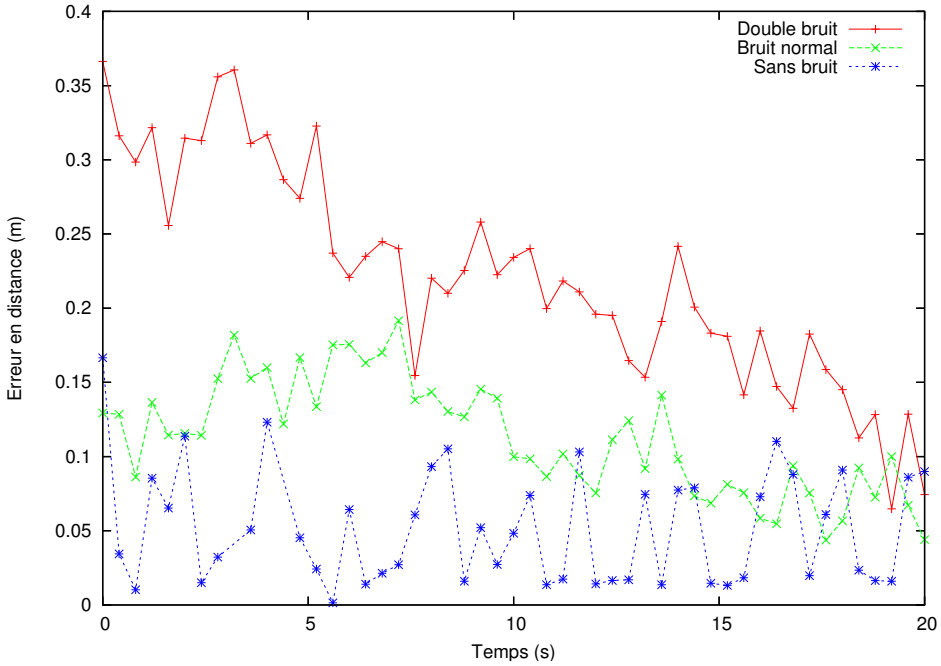


FIG. 6.19 Erreur du point de repère en fonction du bruit

6.4.6 Conclusion

Avec les paramètres de simulation décrits à la section 6.2, l'algorithme de localisation relative mutuelle développé dans le cadre du projet permet d'obtenir la position d'une plate-forme à 10 cm près et son orientation à 4° près, et ce, dans un délai inférieur à 1 s.

Parmi les facteurs étudiés pouvant influencer la qualité de l'estimation produite par l'algorithme, deux facteurs ressortent davantage : la vitesse des plates-formes et la quantité de bruit du système. Le premier influence la variance de l'erreur alors que le second influence plutôt l'erreur moyenne.

Parmi les autres facteurs, le nombre de points de repère ainsi que la qualité de l'estimation de la localisation relative servant à initialiser le filtre de Kalman permettent tous deux à l'algorithme de converger plus ou moins vite vers la localisation réelle.

Le nombre de plates-formes impliquées, quant à lui, semble n'influencer que très peu les résultats.

Enfin, il a été démontré par la simulation qu'un point de repère seul ne suffit pas à l'algorithme pour se comporter convenablement. On note également que le filtre s'adapte suffisamment rapidement pour qu'il ne soit pas nécessaire de développer une autre méthode de localisation relative non récursive permettant d'initialiser le filtre de Kalman.

6.5 Résultats d'expérimentation

Cette section présente les résultats obtenus sur les plates-formes décrites à la sous-section 2.4.1 avec la méthode de localisation relative mutuelle implantée dans le cadre de ce projet de recherche. La scène a été disposée comme à la figure 6.1 afin de pouvoir comparer sur une même base les résultats réels avec les résultats simulés.

Comme expliqué précédemment, afin de calculer l'erreur de l'estimation produite par la méthode, il est nécessaire de localiser de façon absolue les plates-formes et les points de repère. En simulation, *Stage* permettait d'obtenir directement ces informations.

Cependant, les plates-formes réelles ne sont pas munies de dispositifs de localisation absolue. Par conséquent, les mesures de localisation absolue ont été prises manuellement, avec les erreurs de mesure qui les accompagnent. Par conséquent, il nous était impossible de connaître avec certitude l'erreur d'estimation.

De surcroît, plusieurs facteurs matériels peuvent biaiser les résultats obtenus. Par exemple, la méthode de calibrage de distance verticale de la webcam par rapport

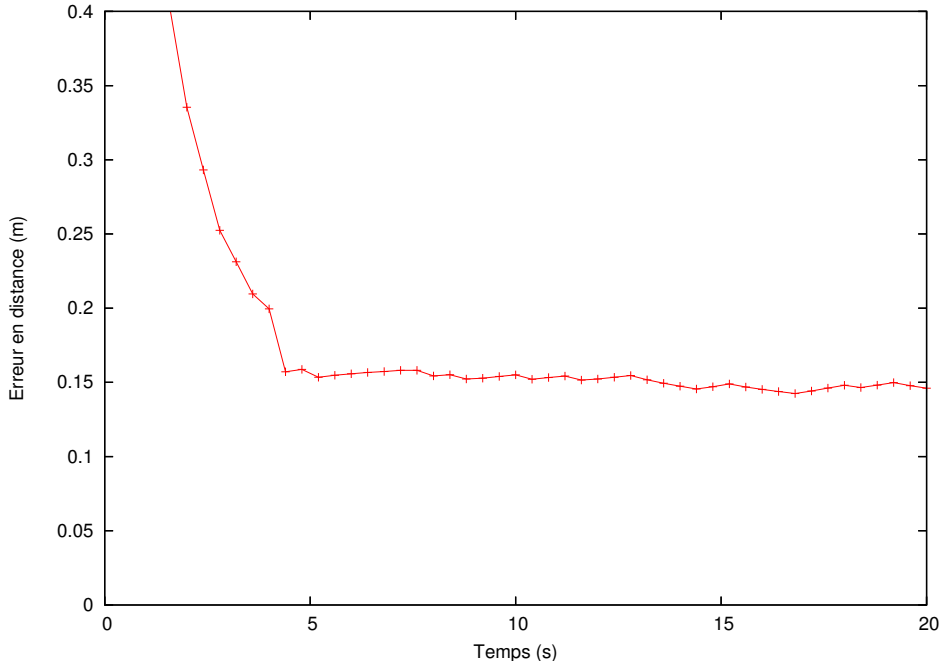


FIG. 6.20 Erreur de position d'une plate-forme

au miroir hyperboloïdal est imprécise et génère des erreurs d'évaluation de distance pour les points de repère. De plus, l'axe optique n'est probablement pas parfaitement aligné avec l'axe de l'hyperboloïde.

6.5.1 Résultats quantitatifs

Comme la localisation absolue ne peut être mesurée que manuellement dans notre cas, l'obtention de résultats quantitatifs ne peut impliquer des plates-formes en mouvement. Par conséquent, les tests qui suivent ont été réalisés avec des plate-formes immobiles.

Les figures 6.20, 6.21 et 6.22 montrent respectivement les erreurs de l'estimation de la position d'une plate-forme, de l'orientation de la plate-forme et de la position d'un point de repère.

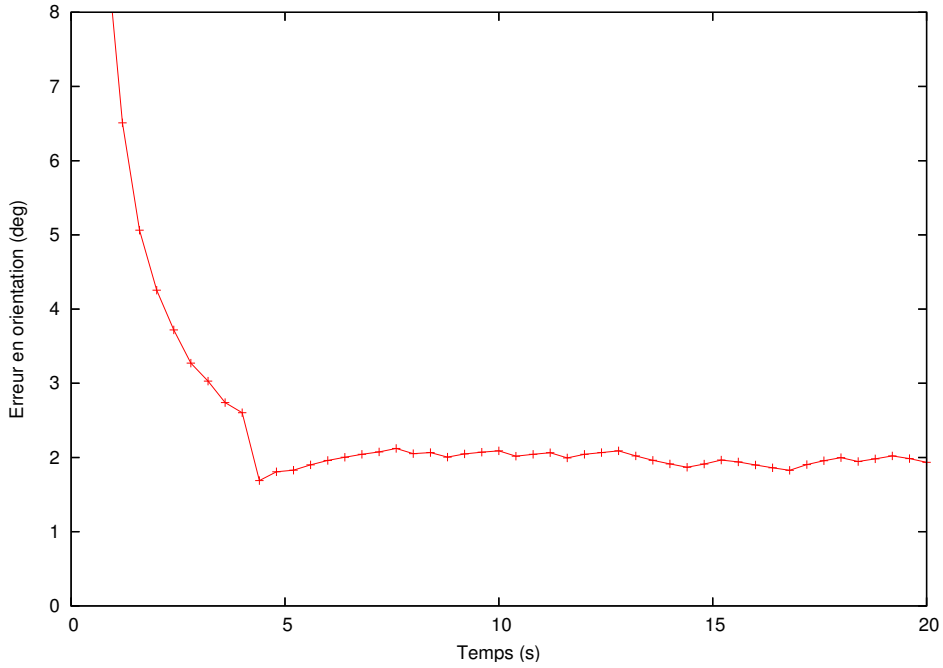


FIG. 6.21 Erreur d'orientation d'une plate-forme

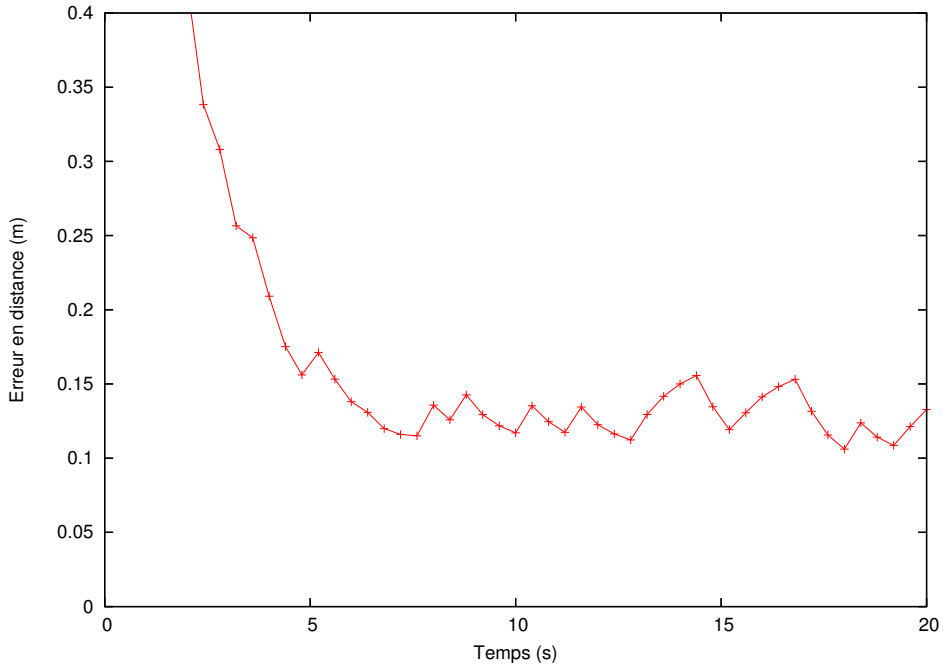


FIG. 6.22 Erreur de position d'un point de repère

Étant donné l'immobilité des plates-formes, le bruit du système est minimal, ce qui permet d'obtenir une faible variance pour l'estimation de la position d'une plate-forme. Après 5 s, l'erreur se stabilise à 15 cm et ne semble plus vouloir converger vers la solution réelle. Cette distance correspond en fait au biais encouru par les divers facteurs matériels tels que discutés précédemment : si les mesures manuelles étaient parfaites, si les caméras omnidirectionnelles étaient parfaitement calibrées, alors l'erreur tomberait probablement en-deça de 5 cm pour deux plates-formes immobiles, avec trois points de repère.

Si l'on compare la figure 6.20 avec la figure 6.2 (qui correspond au même test, mais en simulation), on remarque que le temps de convergence dans la réalité est beaucoup plus long qu'en simulation. La cause principale est l'incertitude accrue des détections effectuées par les caméras omnidirectionnelles. Afin de compenser de plus grandes erreurs dues aux vibrations importantes des plates-formes lors de leurs déplacements, l'implantation sur les robots de l'algorithme a nécessité une augmentation des valeurs de la matrice de covariance du bruit de mesure. Par conséquent, le filtre de Kalman a tendance à faire moins confiance aux mesures des caméras omnidirectionnelles, ce qui explique le délai de convergence plus grand : environ 5 s.

Les mêmes remarques s'appliquent à l'estimation de l'orientation d'une plate-forme : faible variance, biais d'environ 2°, temps de convergence d'environ 5 s. La variance augmente sensiblement dans le cas de l'estimation de la position d'un point de repère. En effet, comme cette valeur dépend directement de la localisation effectuée par la caméra omnidirectionnelle, le bruit de mesure influence davantage les résultats. Après 5 s, l'erreur devient inférieure à 15 cm, ce qui est tout de même acceptable. Cependant, le biais doit être ici d'environ 10 cm, ce qui laisse présager, dans des conditions idéales, une précision de l'ordre de 5 cm.

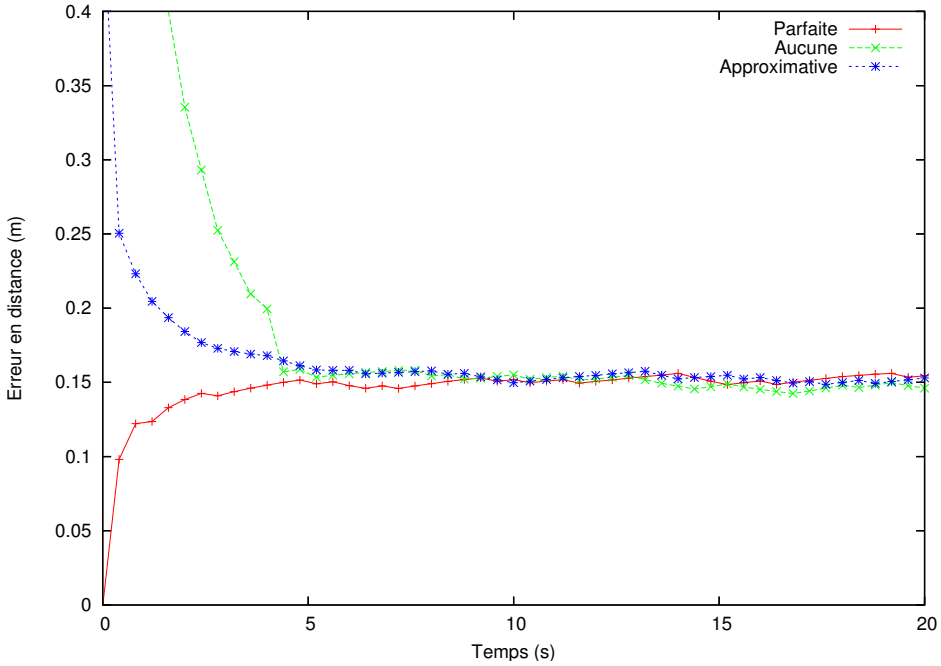


FIG. 6.23 Erreur de position en fonction de la qualité de l'initialisation

L'influence de la qualité de l'initiliasation du filtre de Kalman, tel que discutée à la sous-section 6.4.3, a également été étudiée avec des plates-formes réelles afin de savoir si la méthode développée pouvait se passer d'une méthode non récursive de localisation afin d'initialiser le filtre. Les figures 6.23, 6.24 et 6.25 présentent les résultats de ce test.

Il est évident, par inspection de ces graphiques, que si l'on néglige le biais encouru en raison de facteurs matériels, les estimations convergent beaucoup plus rapidement si l'initialisation du filtre de Kalman est parfaite ou approximative que s'il n'y a aucune initialisation. Il n'en demeure pas moins qu'après 5 s, l'algorithme est en régime permanent même en l'absence d'initialisation convenable.

Un indice supplémentaire prouvant que l'erreur moyenne élevée est due à un biais est le fait que l'erreur commence généralement proche de 0 dans le cas d'une initialisation parfaite et se détériore par la suite pour converger vers la meilleure solution,

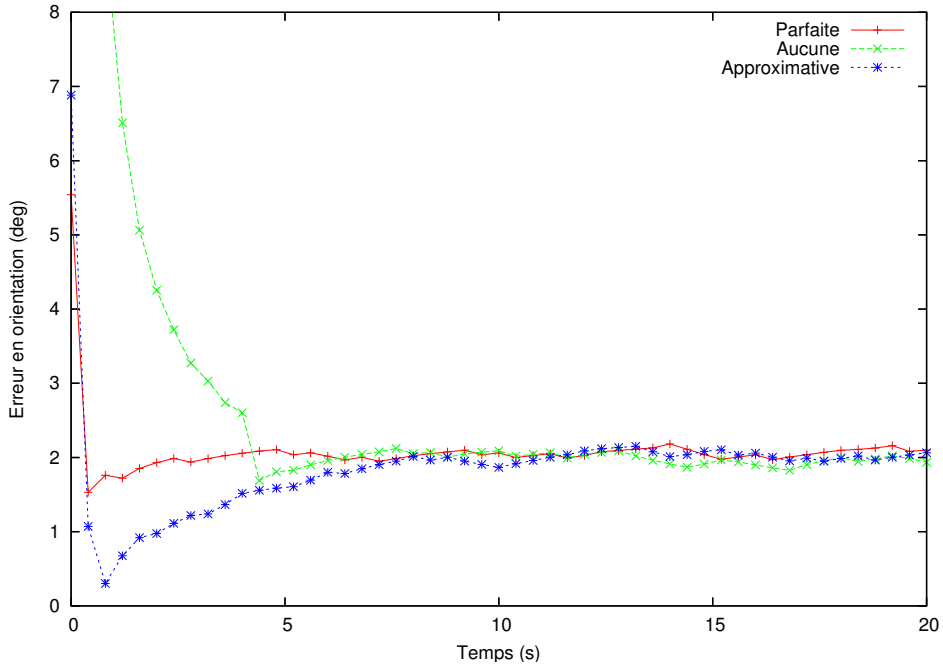


FIG. 6.24 Erreur d'orientation en fonction de la qualité de l'initialisation

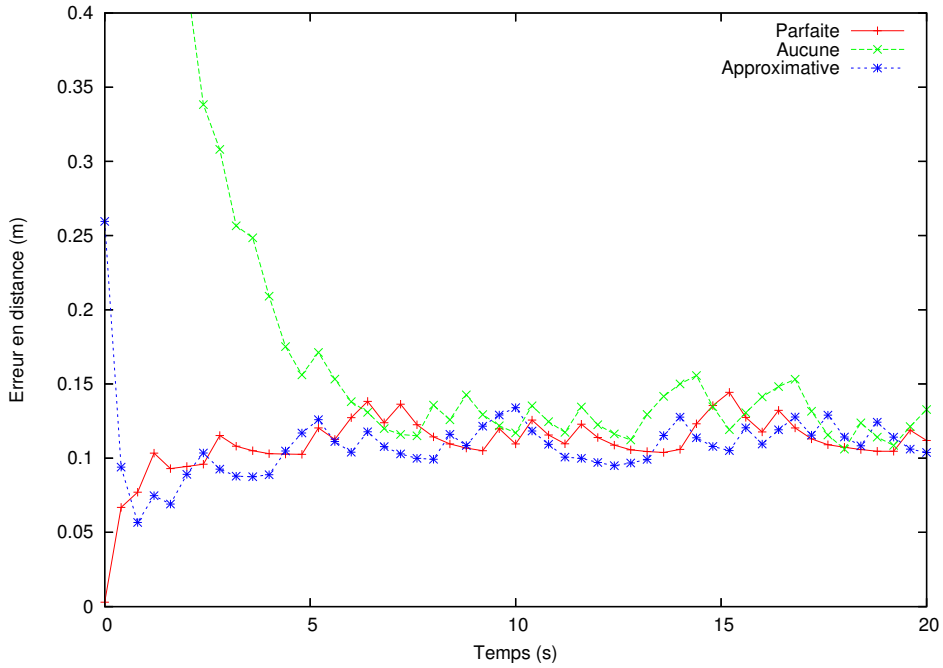


FIG. 6.25 Erreur du point de repère en fonction de la qualité de l'initialisation

compte tenu du biais.

6.5.2 Résultats qualitatifs

Afin de montrer que l'algorithme fonctionne en réalité lorsque les plates-formes se déplacent dans leur environnement, des résultats qualitatifs seront présentés dans cette sous-section, puisque la mesure manuelle de localisation absolue devient impossible.

Le CD-ROM accompagnant ce document contient un extrait vidéo permettant de voir les plates-formes se déplacer autour des points de repère de la scène. En parallèle, une interface graphique affichant le résultat de la localisation relative permet de constater que la méthode donne une excellente approximation de la localisation relative mutuelle, et ce, en temps réel sur un ordinateur doté d'un processeur de 1 GHz (l'ordinateur de la plate-forme *ATRV-Mini*).

En plus de démontrer que la méthode fonctionne, des tests effectués lors de cette expérimentation et visibles dans l'extrait vidéo permettent d'apprécier la robustesse de l'implantation de l'algorithme, telle que décrite à la section suivante.

6.6 Robustesse

Même si la méthode de base fonctionne bien en simulation, il est important d'ajouter de la robustesse à la méthode afin de s'assurer que des événements imprévus ne déstabilisent pas complètement les résultats. Voici des exemples de vérifications effectuées pendant l'exécution du logiciel développé afin de s'assurer un fonctionnement stable.

6.6.1 Nombre de points de repère variable

Lorsque le filtre de Kalman est initialisé, l'état ne comporte la position d'aucun point de repère. En fait, ces points de repère sont ajoutés à l'état au fur et à mesure qu'ils sont détectés.

Par conséquent, un filtre de Kalman à dimension variable a été implanté pour permettre de modifier dynamiquement la taille de l'état, c'est-à-dire le nombre de variables à estimer. Ainsi, l'algorithme pourrait en théorie fonctionner pendant des heures et découvrir de nouveaux points de repère en chemin et les ajouter à l'état.

Afin d'éviter que la taille de l'état ne croisse sans fin, les points de repère qui ne sont plus visibles doivent être éliminés de l'état. Cependant, il ne faut pas qu'un point de repère soit éliminé dès sa disparition. En effet, il arrive fréquemment qu'un point de repère ne soit pas détecté, pendant seulement le temps d'une itération. Le fait de garder tout de même le point de repère dans l'état permet d'avoir déjà une bonne estimation lorsqu'il est détecté à nouveau. Il faut donc n'éliminer le point de repère de l'état que s'il n'a pas été détecté depuis un certain temps, et c'est ce qui a été implanté dans le cadre du projet.

L'extrait vidéo démontre que, si le point de repère n'est pas détecté pendant un temps inférieur à 2 s, il ne disparaît pas de l'interface graphique, c'est-à-dire qu'il est conservé dans l'état et sa position est tout de même mise à jour.

Inversement, l'extrait montre également que si le point de repère disparaît pendant plus de temps, alors sa position est oubliée. Cela ne l'empêche pas de réapparaître plus tard à un autre endroit, auquel cas quelques itérations seront nécessaires pour le situer correctement.

6.6.2 Déplacement des points de repère

Les équations de mise à jour de l'état telles que définies dans le modèle du processus pour le filtre de Kalman supposent que les points de repère sont fixes. Néanmoins, deux variables de bruit de processus ont été ajoutées à la position d'un point de repère afin de lui permettre de se déplacer légèrement et ainsi de corriger plus rapidement les erreurs d'estimation.

Ceci permet au système de s'adapter lentement à des déplacements des points de repère, même si cela n'était pas du tout prévu dans le système. Naturellement, l'estimation en souffre, mais après quelques secondes, elle converge à nouveau vers la localisation relative réelle, comme le démontre l'extrait vidéo.

6.6.3 Communication sans fil

De nos jours, la communication sans fil est assez fiable. Néanmoins, il arrive fréquemment que des délais imprévus surviennent lors de l'utilisation d'une connexion sans fil.

Beaucoup d'efforts ont été investis afin que le système soit robuste à de tels problèmes de réseau. Ainsi, un mécanisme logiciel permet à toutes les plates-formes de se synchroniser lors du passage d'une itération à l'autre. De plus, la connexion peut être interrompue pendant 5 s et être rétablie sans trop heurter les résultats. Néanmoins, si une plate-forme se déconnecte, alors les autres plates-formes s'arrêtent également. Une amélioration envisagée est de donner la possibilité au filtre de Kalman d'accepter des nouvelles plates-formes et d'être robuste aux déconnexions.

6.7 Conclusion

Ce chapitre a démontré la validité de la méthode d'estimation de la localisation relative mutuelle développée dans le cadre de ce projet de recherche. Grâce à la simulation, l'influence de différents facteurs sur la précision de l'estimation a pu être observée. La vitesse des plates-formes et la quantité de bruit du système sont les deux facteurs ayant le plus d'impact sur la variance et la moyenne de l'erreur d'estimation, respectivement.

L'expérimentation avec des plates-formes réelles a montré qu'il était possible d'obtenir de très bon résultats avec cette méthode. De manière plus quantitative, une erreur moyenne d'environ 15 cm sur la localisation de plates-formes immobiles a pu être observée. La cause probable de cette erreur est un biais de l'estimation dû à des facteurs matériels, tels que la précision de mesures manuelles et du calibrage du système catadioptrique.

CONCLUSION

La localisation d'une plate-forme mobile est un problème étudié depuis quelques décennies et qui est encore d'actualité aujourd'hui. Il est important, pour qu'une plate-forme puisse cartographier correctement un environnement ou planifier adéquatement une tâche, qu'elle sache où elle se situe.

La localisation absolue d'une plate-forme est la première étape que les chercheurs ont abordée. Plusieurs solutions à ce problème sont maintenant reconnues et robustes.

L'avenir laisse entrevoir la coopération entre plusieurs plates-formes afin d'accomplir des tâches concertées. Un nouveau besoin émerge de cette réalité : celui de pouvoir localiser les plates-formes les unes par rapport aux autres. Le problème de localisation relative mutuelle est plus récent dans la littérature scientifique. Plusieurs solutions existent déjà, mais pourraient être améliorées ou sont sujettes à des contraintes qu'il serait avantageux d'éliminer.

La méthode de localisation relative mutuelle développée dans le cadre de ce projet de maîtrise permet d'obtenir des résultats de bonne qualité en utilisant du matériel à faible coût. Seuls un système d'odométrie et une caméra omnidirectionnelle sont nécessaires afin d'obtenir une estimation entachée d'une erreur inférieure à 20 cm pour des plates-formes mobiles bien calibrées.

Une autre force de la méthode proposée est sa rapidité d'exécution. Il est important pour une plate-forme d'obtenir une estimation de sa position en temps réel afin qu'elle puisse prendre les décisions qui s'imposent, au bon moment. Or, notre méthode permet d'obtenir en moins de 5 s la configuration initiale des plates-formes sans aucune connaissance *a priori* de l'environnement ou de la position initiale des

plates-formes. La seule contrainte est la possibilité, pour chaque plate-forme, de détecter des points de repère fixes (mais de positions initiales inconnues) et de les apparier avec les détections des autres plates-formes.

En régime permanent, mettre à jour plus de deux fois par seconde la localisation relative des plates-formes est possible avec un ordinateur muni d'un processeur cadencé à 1 GHz, et ce, en consommant généralement moins de 20% des ressources de l'ordinateur.

L'aspect innovateur du projet réside dans le fait que les plates-formes sont capables de se localiser sans connaissance *a priori* de leur environnement et sans même établir de contact visuel direct à aucun moment. Le partage d'informations de détections et de déplacements suffit à permettre d'estimer la localisation relative des plates-formes. Le concept de filtre de Kalman étendu à dimension variable est également une contribution innovatrice de ce travail.

De plus, le système est robuste et flexible : il s'adapte à différents événements, même inattendus. Il est possible d'ajouter et de retrancher des points de repère dynamiquement au cours d'une exploration. Si un point de repère ne peut être détecté pendant un certain temps, sa position sera tout de même mise en jour. Un point de repère, supposé fixe, peut même se déplacer sans perturber exagérément l'algorithme, qui s'adapte généralement au bout de quelques secondes.

Une architecture logicielle évolutive d'intégration et de prototypage rapide a également été développée dans le cadre du projet et a été utilisée pour implanter la méthode de localisation relative mutuelle. Cette architecture pourra servir à augmenter le rendement de futurs chercheurs qui voudraient utiliser conjointement la méthode développée avec d'autres projets de robotique mobile.

Le système développé comprend quelques faiblesses. Le système de vision utilisé est

embryonnaire : pour simplifier l'implantation de la solution, seuls des cylindres à la verticale et uniformément colorés sont détectés dans les images omnidirectionnelles. Un système réaliste de localisation relative mutuelle devrait comporter un système de vision capable de se baser sur des points de repère naturels de l'environnement. Il s'agit là d'une voie de recherche en soi, et les chercheurs investissent des efforts considérables afin d'atteindre cet objectif.

La contrainte fondamentale de la méthode suppose que différentes plates-formes soient capables de mettre en correspondance les points de repère qu'elles ont détectés. Dans le cas simplifié, la couleur permet d'identifier de façon unique les balises cylindriques. En supposant un système de vision capable de détecter des points de repère naturels, le problème de mise en correspondance jumelé à la possibilité que plusieurs points de repère aient une apparence identique vient complexifier le problème. Cet aspect constitue une amélioration importante du système afin de se rapprocher d'un système concret et réaliste de localisation relative mutuelle.

La méthode telle qu'implantée dans le cadre du projet suppose que le sol est plat et que les points de repère entre en contact avec le sol afin de pouvoir non seulement estimer la direction où se trouve le point de repère, mais également approximer la distance qui le sépare de la plate-forme l'ayant détecté. Il s'agit d'une contrainte sévère qui peut facilement être éliminée en utilisant la stéréographie. En effet, deux caméras omnidirectionnelles peuvent être installées sur chaque plate-forme afin de pouvoir déterminer avec suffisamment de précision la position d'un point de repère sans émettre ces hypothèses contraignantes.

Enfin, quelques idées pourraient être exploitées afin d'améliorer les performances et les capacités de la méthode. Une de ses forces réside dans le fait que le contact visuel entre les plates-formes n'est pas nécessaire. Toutefois, rien n'empêche que les plates-formes capables de détecter certaines des autres plates-formes devraient

utiliser ces informations supplémentaires afin d'améliorer davantage la qualité de l'estimation de la localisation relative mutuelle. Cela revient à dire que les plates-formes pourraient également agir à titre de points de repère mobiles, mais dont on peut connaître la trajectoire. Ce nouveau type de points de repère pourrait être intégré au filtre de Kalman utilisé dans le projet afin de rendre le système encore plus performant.

Le système devrait également avoir la possibilité de supporter la connexion et la déconnexion dynamique de certaines plates-formes afin d'être plus flexible. Ainsi, le filtre de Kalman à dimension variable devrait également modifier la taille de l'état pour prendre en compte l'arrivée d'une nouvelle plate-forme, par exemple.

En conclusion, le projet a permis de développer le prototype d'un système concret, temps réel de localisation relative mutuelle se basant sur la vision pour détecter des points de repère de position initiale inconnue, sans connaissance *a priori* de l'environnement. Les développements futurs, particulièrement au niveau du système de vision, permettront de construire un système fiable de localisation qui servira de bloc de base aux algorithmes décisionnels de la robotique mobile de l'avenir.

RÉFÉRENCES

Albus, J.S.. 1992. « A reference model architecture for intelligent systems design ». *An Introduction to Intelligent and Autonomous Control*. Kluwer Academic Publishers. P. 57–64.

Andersson, M., Oreback, A., Lindstrom, M., and Christensen, H.I.. 1998. « Isr : An intelligent service robot ». *Sensor Based Intelligent Robots ; International Workshop Dagstuhl Castle*.

Baker, Simon and Nayar, Shree K.. 1999. « A theory of single-viewpoint catadioptric image formation ». *International Journal of Computer Vision*. Kluwer Academic Publishers. P. 175–196.

Barth, Matthew J. and Ishiguro, Hiroshi. 1994. « Distributed panoramic sensing in multiagent robotics ». *Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*. Las Vegas : Institute of Electrical and Electronics Engineers Inc. P. 739–746.

Bierman, Gerald J.. 1977. *Factorization methods for discrete sequential estimation*. New York : Academic Press. 241p.

Brooks, Alex, Makarenko, Alexei, Kaupp, Tobias, Williams, Stefan, and Durrant-Whyte, Hugh. 2004. « Implementation of an indoor active sensor network ». *9th Int. Symp. on Experimental Robotics (ISER ?04)*.

Chatila, R. and Laumond, J.P.. 1985. « Position referencing and consistant world modeling for mobile robots ». *Proceedings IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc.

Choset, Howie and Nagatani, Keiji. 2001. « Topological simultaneous localization and mapping (slam) : toward exact localization without explicit localiza-

tion ». *IEEE Transactions on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc. P. 125–137.

Clerentin, Arnaud, Delahoche, Laurent, Brassart, Eric, and Pegard, Claude. 2001. « Omnidirectional sensors cooperation for multi-target tracking ». *International Conference on Multisensor Fusion and Integration for Intelligent Systems*. Institute of Electrical and Electronics Engineers Inc. P. 335–340.

Clerentin, Arnaud, Delahoche, Laurent, Pegard, Claude, and Brassart, Eric. 2000. « A localization method based on two omnidirectional perception systems cooperation ». *Proceedings of the 2000 IEEE International Conference on Robotics & Automation*. San Francisco : Institute of Electrical and Electronics Engineers Inc. P. 1219–1224.

Côté, C., Létourneau, D., Michaud, F., Valin, J.-M., Brosseau, Y., Raïevsky, C., Lemay, M., and Tran, V.. 2004. « Code reusability tools for programming mobile robots ». *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers Inc. P. 1820–1825.

Cullen, Jennings, Murray, Don, and Little, James J.. 1999. « Cooperative robot localization with vision-based mapping ». *Proceedings of the 1999 IEEE International Conference on Robotics & Automation*. Detroit : Institute of Electrical and Electronics Engineers Inc. P. 2659–2665.

Dellaert, Frank and Stroupe, Ashley W.. 2002. « Linear 2d localization and mapping for single and multiple robot scenarios ». *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*. Washington : Institute of Electrical and Electronics Engineers Inc. P. 688–694.

di Marco, Mauro, Garulli, Andrea, Giannitrapani, Antonio, and Vicino, Antonio. 2003. « Simultaneous localization and map building for a team of coope-

rating robots : A set membership approach ». *IEEE Transactions on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc. P. 238–249.

Dissanayake, M.W.M. Gamini, Newman, Paul, Clark, Steven, Durrant-Whyte, Hugh F., and Csorba, M.. 2001. « A solution to the simultaneous localization and map building (slam) problem ». *IEEE Transactions on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc. P. 229–241.

Doncarli, C., Le Corre, J.F., and Devise, O.. 1991. « Dynamic location of a mobile robot by extended kalman filtering ». *Proceedings IROS '91. IEEE/RSJ International Workshop on Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems*. Osaka : Institute of Electrical and Electronics Engineers Inc. P. 1433–1435.

Drocourt, Cyril, Delahoche, Laurent, Pegard, Claude, and Cauchois, Cyril. 1999. « Localization method based on omnidirectional stereoscopic vision and dead-reckoning ». *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers Inc. P. 960–965.

Fleury, S., Herrb, M., and Chatila, R.. 1997. « Genom : a tool for the specification and the implementation of operating modules in a distributed robot architecture ». *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots & Systems (IROS)*. Grenoble : Institute of Electrical and Electronics Engineers Inc. P. 842–848.

Gerkey, B., Vaughan, R., and Howard, A.. 2003. « The player/stage project : Tools for multi-robot and distributed sensor systems ». *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*. P. 317–323.

Gerkey, Brian, Vaughan, Richard, Howard, Andrew, and Koenig, Nate. 2005. « The player/stage project ». [En ligne]. http ://players-tage.sourceforge.net/ (Consulté le 2 octobre 2005).

Ikeda, Sei, Sato, Tomokazu, and Yokoya, Naokazu. 2003. « High-resolution panoramic movie generation from video streams acquired by an omnidirectional multi-camera system ». *Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI2003*. Institute of Electrical and Electronics Engineers Inc. P. 155–160.

Kalman, R.E. and Bucy, R.. 1960. « A new approach to linear filtering and prediction problems ». *Transactions ASME, Journal of Basic Engineering*.

Kato, Koji, Ishiguro, Hiroshi, and Barth, Matthew. 1999. « Identifying and localizing robots in a multi-robot system environment ». *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers Inc. P. 966–971.

Konolige, Kurt and Myers, K.. 1998. « The saphira architecture for autonomous mobile robots ». *Artificial Intelligent and Mobile Robots*. AAAI Press. P. 211–242.

Lallement, A., Siadat, A., Dufaut, M., and Husson, R.. 1998. « Laser-vision cooperation for map building and landmarks recognition ». *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*. Gaithersburg : Institute of Electrical and Electronics Engineers Inc. P. 387–392.

Madhavan, Raj, Durrant-Whyte, Hugh, and Dissanayake, Gamini. 2002. « Natural landmark-based autonomous navigation using curvature scale space ». *Proceedings of the 2002 IEEE International Conference on Robotics & Automation*. Washington : Institute of Electrical and Electronics Engineers Inc. P. 3936–3941.

Micusik, Branislav and Pajdla, Tomas. 2003. « Estimation of omnidirectional camera model from epipolar geometry ». *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Institute of Electrical and Electronics Engineers Inc. P. 485.

More, J.J.. 1977. *The Levenberg-Marquardt algorithm : Implementation and theory*. Numerical Analysis.

Nayar, Shree K.. 1997. « Catadioptric omnidirectional camera ». *1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*. Institute of Electrical and Electronics Engineers Inc. P. 482.

Nüchter, Andreas, Lingemann, Kai, Hertzberg, Joachim, and Surmann, Hartmut. 2005. « Accurate object localization in 3d laser range scans ». *Proceedings of the International Workshop on Safety, Security and Rescue Robotics*.

Rekleitis, Ioannis, Sim, Robert, Dudek, Gregory, and Milios, Evangelos. 2001. « Collaborative exploration for map construction ». *Proceedings of the 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*. Banff : Institute of Electrical and Electronics Engineers Inc. P. 296–301.

Rosenblatt, J.K. and Thorpe, C.E.. 1995. « Combining multiple goals in a behavior-based architecture ». *International Conference on Intelligent Robots and Systems*. Institute of Electrical and Electronics Engineers Inc. P. 136.

Spletzer, J., Das, A. K., Fierro, R., Taylor, C. J., Kumar, V., and Ostrowski, J. P.. 2001. « Cooperative localization and control for multi-robot manipulation ». *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Hawaii : Institute of Electrical and Electronics Engineers Inc. P. 631–636.

Thrun, Sebastian, Burgard, Wolfram, and Fox, Dieter. 2000. « A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping ». *Proceedings of the 2000 IEEE International Conference on Robotics & Automation*. San Francisco : Institute of Electrical and Electronics Engineers Inc. P. 321–328.

Thrun, Sebastian, Liu, Yufeng, Koller, Daphne, Y. Ng, Andrew, Ghahramani, Zoubin, and Durrant-Whyte, Hugh. 2004. « Simultaneous localization

and mapping with sparse extended information filters ». *The International Journal of Robotics Research*. SAGE Publications. P. 693–716.

WELCH, Greg and BISHOP, Gary. 2003. « An introduction to the kalman filter ». [En ligne], http ://www.cs.unc.edu/∼welch/media/pdf/kalman_intro.pdf. (Consulté le 2 octobre 2005).

WILLS, L., KANNAN, S., SANDER, S., GULER, M., HECK, B., PRASAD, V., SCHRAGE, D., and VACHTSEVANOS, G.. 2001. « An open platform for reconfigurable control ». Institute of Electrical and Electronics Engineers Inc. P. 49–64.

YAMAZAWA, Kazumasa, YAGI, Yasushi, and YACHIDA, Masahiko. 1993. « Omnidirectional imaging with hyperboloidal projection ». *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems '93, IROS '93*. Yokohama : Institute of Electrical and Electronics Engineers Inc. P. 1029–1034.

ANNEXE I

PROJECTION AVEC MIROIR HYPERBOLOÏDAL

Cette annexe démontre l'équation (4.3) de projection d'un point de scène pour un système catadioptrique utilisant un miroir hyperboloïdal. Pour faciliter les calculs, la convention suivant laquelle le plan image se trouve en avant du centre de projection sera utilisée.

Soit l'hyperboloïde H avec ses foyers $F_1 = (0, 0, 0)$ et $F_2 = (0, 0, -2c)$. Soit le point de la scène $P_1 = (R, \theta, Z)$. Soit le plan image $\Pi : Z = f - 2c$. Le point sur le plan Π résultant de la projection de P_1 est $P_2 = (r, \theta, f - 2c)$. L'objectif est de déterminer r. La figure I.1 présente la situation.

Le point P_1 est tout d'abord projeté en un point $P^* = (R^*, \theta, Z^*)$ sur l'hyperboloïde H en suivant la droite D_1 reliant P_1 et le foyer F_1. Grâce à la propriété énoncée à la sous-section 4.3.2, nous savons que le point P^* est projeté au point P_2 en suivant la droite D_2 qui relie P^* au foyer F_2.

Le point P^* est donc l'intersection de H et D_1 :

$$\begin{bmatrix} R^* \\ Z^* \end{bmatrix} = t \begin{bmatrix} R \\ Z \end{bmatrix} \tag{I.1}$$

$$\frac{(Z^* + c)^2}{a^2} - \frac{(R^*)^2}{c^2 - a^2} = 1 \tag{I.2}$$

L'équation (4.2) a été utilisée pour éliminer b de l'équation (4.1).

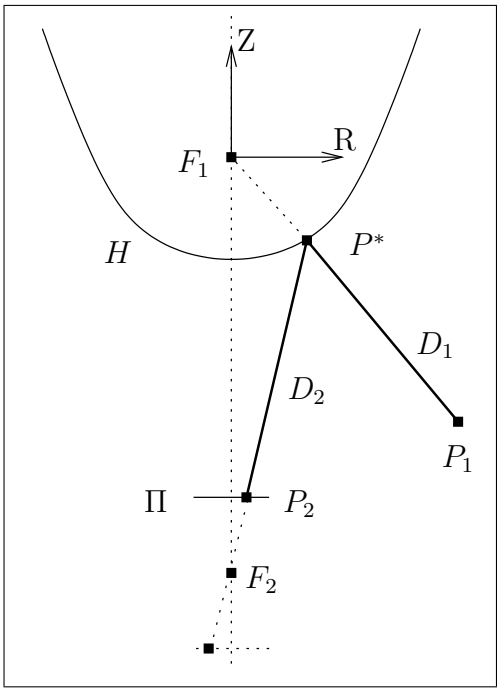


FIG. I.1 Projection d'un point de scène sur le miroir hyperboloïdal

Le point P_2 est l'intersection de D_2 et Π :

$$\begin{bmatrix} r \\ f-2c \end{bmatrix} = \begin{bmatrix} 0 \\ -2c \end{bmatrix} + t' \begin{bmatrix} R^* \\ Z^*+2c \end{bmatrix} \tag{I.3}$$

Il est possible d'isoler r dans (I.3) en éliminant t' :

$$\frac{r}{R^*} = \frac{f-2c-(-2c)}{Z^*+2c} \tag{I.4}$$

$$r = \frac{fR^*}{Z^*+2c} \tag{I.5}$$

Pour éliminer Z^*, on l'isole dans (I.1) :

$$Z^* = R^* \frac{Z}{R} \tag{I.6}$$

$$\begin{aligned} r &= \frac{fR^*}{R^* \frac{Z}{R} + 2c} \\ &= \frac{fR}{Z + \frac{2cR}{R^*}} \end{aligned} \tag{I.7}$$

Pour éliminer R^*, il est nécessaire de l'isoler dans (I.2) en substituant Z^* grâce à (I.6) :

$$\frac{\left(R^* \frac{Z}{R} + c\right)^2}{a^2} - \frac{(R^*)^2}{c^2 - a^2} = 1 \tag{I.8}$$

Il faut résoudre une équation du second degré :

$$(R^*)^2 \left(\left(\frac{Z}{Ra} \right)^2 - \frac{1}{c^2 - a^2} \right) + R^* \left(\frac{2cZ}{Ra^2} \right) + \left(\frac{c^2}{a^2} - 1 \right) = 0 \tag{I.9}$$

$$R^* = \frac{R\left(c^2 - a^2\right)\left(cZ \pm a\sqrt{R^2 + Z^2}\right)}{-c^2 Z^2 + a^2\left(R^2 + Z^2\right)} \tag{I.10}$$

Posons $\epsilon = \pm 1$. Le dénominateur de (I.10) est une différence de carrés. Ceci permet de simplifier l'équation davantage :

$$\begin{aligned} R^* &= \frac{R\left(c^2 - a^2\right)\left(cZ + \epsilon a\sqrt{R^2 + Z^2}\right)}{\left(cZ + \epsilon a\sqrt{R^2 + Z^2}\right)\left(-cZ + \epsilon a\sqrt{R^2 + Z^2}\right)} \\ &= \frac{R\left(c^2 - a^2\right)}{-cZ + \epsilon a\sqrt{R^2 + Z^2}} \end{aligned} \tag{I.11}$$

Il reste à substituer (I.11) dans (I.7) :

$$\begin{aligned} r &= \frac{fR}{Z + 2cR\frac{-cZ+\epsilon a\sqrt{R^2+Z^2}}{R(c^2-a^2)}} \\ &= \frac{(c^2-a^2)\,fR}{(c^2-a^2)\,Z - 2c^2Z + 2ac\epsilon\sqrt{R^2+Z^2}} \\ &= \frac{(c^2-a^2)\,fR}{-(c^2+a^2)\,Z + 2ac\epsilon\sqrt{R^2+Z^2}} \end{aligned} \tag{I.12}$$

En prenant $\epsilon = +1$ (ce qui sera justifié par la suite) et en utilisant les définitions de l'équation (4.3), on obtient le résultat désiré :

$$r = \frac{q_1 fR}{-q_2 Z + q_3\sqrt{R^2+Z^2}} \tag{I.13}$$

où :

$$q_1 = c^2 - a^2, \quad q_2 = c^2 + a^2, \quad q_3 = 2ac \tag{I.14}$$

Justifions maintenant la valeur de ϵ. En coordonnées cylindriques, la valeur de rayon est positive par convention, ce qui veut dire que r, R et R^* sont positifs. De plus, la distance focale f est également définie positive. Enfin, la valeur $(c^2 - a^2)$ est en fait b^2, qui est manifestement positive. Par conséquent, le numérateur de (I.12) est positif. Or, il faut que cette fraction soit positive pour respecter le fait que r doit être positif. On déduit donc que le dénominateur doit être positif :

$$\begin{aligned} -\left(c^2+a^2\right)Z + 2ac\epsilon\sqrt{R^2+Z^2} &> 0 \\ \epsilon &> \frac{(c^2+a^2)\,Z}{2ac\epsilon\sqrt{R^2+Z^2}} \end{aligned} \tag{I.15}$$

Il semble donc que $\epsilon = +1$ soit plus apte à respecter cette inéquation. Puisque les

paramètres a et c sont forcément positifs, cette relation ne pourrait être fausse que si Z est positif et suffisamment grand, plus précisément :

$$\frac{Z}{\sqrt{R^2+Z^2}} \overset{?}{<} \frac{2ac}{(c^2+a^2)} \tag{I.16}$$

La partie gauche de la relation (I.16) est en fait le cosinus de la colatitude du rayon incident. Or, celui-ci ne peut être plus grand que le cosinus de la colatitude de l'asymptote de l'hyperbole. On obtient :

$$\frac{a}{c} \overset{?}{<} \frac{2ac}{(c^2+a^2)} \tag{I.17}$$

$$\frac{2ac}{2c^2} \overset{?}{<} \frac{2ac}{(c^2+a^2)} \tag{I.18}$$

Comme $c^2 > a^2$ selon la relation (4.2), l'inéquation (I.18) est vérifiée, ce qui prouve que $\epsilon = +1$ est une solution valide.

ANNEXE II

ALGORITHME DE SÉLECTION DE PLAGE

II.1 Calcul de moindres carrés

Soit l'ensemble des pixels $\mathbf{p}_i = (x_i, y_i)$, $i = 1, \ldots, N$ d'une plage et $\mathbf{p}_0 = (x_0, y_0)$ le pixel du centre optique de l'image omnidirectionnelle. La demi-droite D de vecteur directeur (unitaire) $\mathbf{n}$ commençant à $\mathbf{p}_0$ et minimisant à la fois la somme des carrés des erreurs des $\mathbf{p}_i$ à la demi-droite peut être exprimée sous forme paramétrique :

$$\mathbf{d}(t) = \mathbf{p}_0 + t\mathbf{n} \tag{II.1}$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + t \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, \qquad t \geq 0 \tag{II.2}$$

L'erreur E_i d'un point $\mathbf{p}_i$ à la demi-droite D est la plus courte distance de ce point à la demi-droite. Comme on peut le voir à la figure II.1, l'erreur peut s'exprimer ainsi :

$$E_i^2 = \|\mathbf{q}_i\|^2 - \|\mathbf{proj}_{\mathbf{n}}\mathbf{q}_i\|^2 \tag{II.3}$$

où :

$$\mathbf{q}_i = [\check{x}_i \;\; \check{y}_i]^{\mathrm{T}} = \mathbf{p}_i - \mathbf{p}_0 \tag{II.4}$$

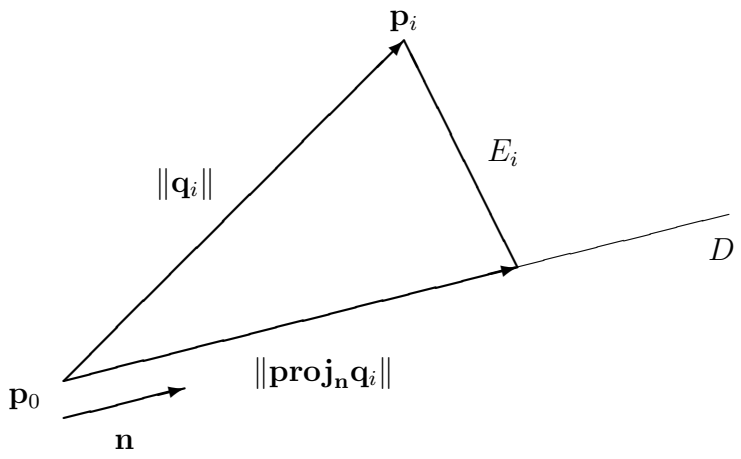


FIG. II.1 Erreur d'un point à la demi-droite

L'expression du carré de l'erreur peut être réorganisée :

$$E_i^2 = \mathbf{q}_i \cdot \mathbf{q}_i - (\mathbf{q}_i \cdot \mathbf{n})^2 \tag{II.5}$$

$$= \check{x}_i^2 + \check{y}_i^2 - \left(\check{x}_i^2 \cos^2\theta + \check{y}_i^2 \sin^2\theta + 2\check{x}_i\check{y}_i \cos\theta \sin\theta\right) \tag{II.6}$$

$$= \check{x}_i^2 \left(1 - \cos^2\theta\right) + \check{y}_i^2 \left(1 - \sin^2\theta\right) - 2\check{x}_i\check{y}_i \cos\theta \sin\theta \tag{II.7}$$

$$= \check{x}_i^2 \sin^2\theta + \check{y}_i^2 \cos^2\theta - 2\check{x}_i\check{y}_i \cos\theta \sin\theta \tag{II.8}$$

$$= (\check{x}_i \sin\theta - \check{y}_i \cos\theta)^2 \tag{II.9}$$

Pour avoir la meilleure droite, il faut minimiser la somme des carrés des erreurs :

$$S = \sum_{i=1}^{N} E_i^2$$

$$= \sum_{i=1}^{N} (\check{x}_i \sin\theta - \check{y}_i \cos\theta)^2 \tag{II.10}$$

La dérivée de S doit donc être nulle :

$$0 = \sum_{i=1}^{N} 2\left(\check{x}_i \sin\theta - \check{y}_i \cos\theta\right)\left(\check{x}_i \cos\theta + \check{y}_i \sin\theta\right) \tag{II.11}$$

$$= \sum_{i=1}^{N} \left[\left(\check{x}_i^2 - \check{y}_i^2\right)\sin\theta\cos\theta + \check{x}_i\check{y}_i\left(\sin^2\theta - \cos^2\theta\right)\right] \tag{II.12}$$

$$= \left[\sum_{i=1}^{N} \frac{\check{x}_i^2 - \check{y}_i^2}{2}\right]\sin 2\theta - \left[\sum_{i=1}^{N} \check{x}_i\check{y}_i\right]\cos 2\theta \tag{II.13}$$

Grâce à ce résultat, on obtient l'angle de la droite :

$$\theta = \frac{1}{2}\arctan\left(\frac{K_2}{K_1}\right) \tag{II.14}$$

avec :

$$K_1 = \sum_{i=1}^{N} \frac{\left(x_i - x_0\right)^2 - \left(y_i - y_0\right)^2}{2} \tag{II.15}$$

$$= \frac{X - Y}{2}$$

$$K_2 = \sum_{i=1}^{N} \left(x_i - x_0\right)\left(y_i - y_0\right) \tag{II.16}$$

où :

$$X = \sum_{i=1}^{N} \left(x_i - x_0\right)^2 \tag{II.17}$$

$$Y = \sum_{i=1}^{N} \left(y_i - y_0\right)^2 \tag{II.18}$$

Afin de réduire la quantité de calculs, il est possible de réécrire les équations (II.16),

(II.17) et (II.18) ainsi :

$$X = X_2 - 2x_0X_1 + Nx_0^2 \tag{II.19}$$

$$Y = Y_2 - 2y_0Y_1 + Ny_0^2 \tag{II.20}$$

$$K_2 = W - y_0X_1 - x_0Y_1 + Nx_0y_0 \tag{II.21}$$

avec :

$$\begin{aligned} X_1 &= \sum_{i=1}^{N} x_i, \quad X_2 = \sum_{i=1}^{N} x_i^2, \\ Y_1 &= \sum_{i=1}^{N} y_i, \quad Y_2 = \sum_{i=1}^{N} y_i^2, \quad W = \sum_{i=1}^{N} x_iy_i \end{aligned} \tag{II.22}$$

Ainsi, lors de la seule traversée des pixels, il suffit de tenir à jour, pour chaque plage, les sommes présentées aux équations (II.22), qui sont simples à calculer. Il est également possible d'exprimer S avec ces mêmes sommes et l'angle θ que l'on vient de calculer :

$$S = \sum_{i=1}^{N} \left(\check{x}_i \sin\theta - \check{y}_i \cos\theta\right)^2 \tag{II.23}$$

$$= \sin^2\theta \sum_{i=1}^{N} \check{x}_i^2 + \cos^2\theta \sum_{i=1}^{N} \check{y}_i^2 - 2\sin\theta\cos\theta \sum_{i=1}^{N} \check{x}_i\check{y}_i \tag{II.24}$$

$$= X\sin^2\theta + Y\cos^2\theta - 2K_2\sin\theta\cos\theta \tag{II.25}$$

II.2 Critère de qualité

Le critère Q devrait être grand pour une plage filiforme (S petit) et devrait également être proportionnel au nombre de pixels N d'une plage afin de privilégier les plus grandes plages. Or, la valeur S croît avec le nombre de pixels N d'une plage.

Il convient donc de diviser S par N afin d'obtenir l'écart carré moyen d'un pixel de la plage à la demi-droite. Cette nouvelle valeur est intimement liée à la largeur moyenne d'une plage autour de la demi-droite. Ainsi, plus ce quotient est faible, plus la plage est filiforme.

À la lumière des conclusions précédentes, il est possible d'énoncer les relations de proportionalité suivantes :

$$\left.\begin{array}{ll} Q \propto \left(\dfrac{S}{N}\right)^{-\alpha} & \alpha > 0 \\ Q \propto N^{\beta} & \beta > 0 \end{array}\right\} \Longrightarrow Q \propto \frac{N^{\alpha+\beta}}{S^{\alpha}} \tag{II.26}$$

Les valeurs $\alpha = 1$ et $\beta = 1.5$ ont été déterminées empiriquement pour notre projet.

ANNEXE III

ÉTALONNAGE DES BALISES

III.1 Calibrage

La procédure consiste à superposer à l'image retournée par la webcam un patron standard montré à la figure III.1. Le miroir étant en fait une hyperboloïde tronquée, il possède un diamètre maximal. Le cercle correspondant à la projection des contours du miroir dans l'image devrait être situé à un endroit bien précis de l'image si la webcam est réellement située au second foyer de l'hyperboloïde.

Pour ajuster la position verticale de la webcam, il faut faire correspondre le contour du miroir avec le grand cercle du patron. Le point central peut aider également à cette tâche : il devrait arriver au centre de la réflexion de la webcam dans le miroir.

Une fois la position verticale ajustée, il est possible de mettre au point la webcam afin d'obtenir des contours plus clairs. Grâce au même programme de calibrage, il est également possible d'ajuster manuellement le gain, le temps d'exposition, les gains de bleu et de rouge, le contraste et d'autres paramètres de la webcam afin d'obtenir une image de meilleure qualité. Tous ces paramètres sont ensuite sauvegardés pour les prochaines utilisations.

L'étape suivante consiste à placer dans la scène les balises cylindriques afin que les rectangles du patron soient entièrement contenus dans la projection des balises. Le programme construit ensuite automatiquement les intervalles de couleur nécessaires à la détection en suivant l'algorithme décrit ci-après, puis sauvegarde ces informations.

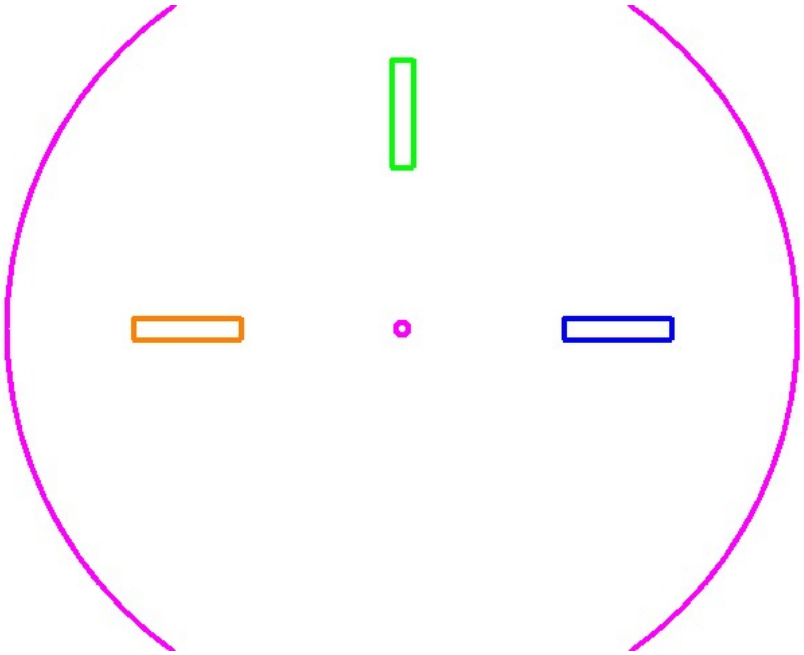


FIG. III.1 Patron de calibrage superposé à l'image omnidirectionnelle

III.1.1 Espace de couleur HSV et construction d'histogrammes

L'image est tout d'abord convertie dans l'espace de couleur HSV, comme expliqué à la sous-section 4.5.1.1. Ensuite chaque région d'intérêt rectangulaire du patron de la figure III.1 est analysée séparément.

Pour connaître la distribution en H, S et V du point de repère, trois histogrammes sont construits à partir des pixels de la région d'intérêt. Par exemple, l'histogramme en H donne, pour chaque valeur de H possible, le nombre de pixels de la région d'intérêt qui ont cette valeur.

III.1.2 Seuillage et marges de sécurité

Il est possible que certains pixels de la région d'intérêt ne soient pas de la couleur du point de repère, que ce soit en raison du bruit dans l'image, de marques sur

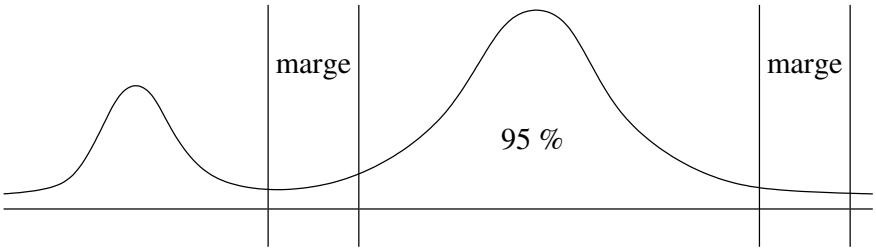


FIG. III.2 Seuillage d'un histogramme

les balises ou parce que la région d'intérêt n'est pas complètement contenue par la projection d'une balise. Dans ce cas, il faut éliminer ces pixels.

Du point de vue d'un histogramme, il s'agit d'établir un poucentage des pixels à conserver et de trouver le plus court intervalle permettant d'obtenir ce nombre de pixels. La figure III.2 montre un exemple de cette procédure de seuillage d'un histogramme.

Une fois le plus court intervalle trouvé, il est plus sécuritaire de lui ajouter une certaine marge de sécurité afin de permettre la détection d'un point de repère dans des conditions légèrement différentes de celles utilisées pour le calibrage, sans avoir à refaire la procédure. La marge de sécurité est moins grande pour la teinte que pour la saturation ou la valeur, car c'est surtout la teinte qui sert à identifier un point de repère. Sinon, il serait plus probable de classer comme point de repère un objet qui n'en est pas un.